

PGP Command Line

User's Guide

Version 7.0

Copyright © 1990-2000 Networks Associates Technology, Inc. All Rights Reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form or by any means without the written permission of Networks Associates Technology, Inc., or its suppliers or affiliate companies.

PGP* Command Line, Version 7.0.1

12-2000. Printed in the United States of America.

LICENSE AGREEMENT

NOTICE TO ALL USERS: FOR THE SPECIFIC TERMS OF YOUR LICENSE TO USE THE SOFTWARE THAT THIS DOCUMENTATION DESCRIBES, CONSULT THE README.1ST, LICENSE.TXT, OR OTHER LICENSE DOCUMENT THAT ACCOMPANIES YOUR SOFTWARE, EITHER AS A TEXT FILE OR AS PART OF THE SOFTWARE PACKAGING. IF YOU DO NOT AGREE TO ALL OF THE TERMS SET FORTH THEREIN, DO NOT INSTALL THE SOFTWARE. IF APPLICABLE, YOU MAY RETURN THE PRODUCT TO THE PLACE OF PURCHASE FOR A FULL REFUND.

NETWORK ASSOCIATES TRADEMARK ATTRIBUTIONS

* *ActiveHelp, Bomb Shelter, Building a World of Trust, CipherLink, Clean-Up, Cloaking, CNX, Compass 7, CyberCop, CyberMedia, Data Security Letter, Discover, Distributed Sniffer System, Dr Solomon's, Enterprise Secure Cast, First Aid, ForceField, Gauntlet, GMT, GroupShield, HelpDesk, Hunter, ISDN Tel/Scope, LM 1, LANGuru, Leading Help Desk Technology, Magic Solutions, MagicSpy, MagicTree, Magic University, MagicWin, MagicWord, McAfee, McAfee Associates, MoneyMagic, More Power To You, Multimedia Cloaking, NetCrypto, NetOctopus, NetRoom, NetScan, Net Shield, NetShield, NetStalker, Net Tools, Network Associates, Network General, Network Uptime!, NetXRay, Nuts & Bolts, PC Medic, PCNotary, PGP, PGP (Pretty Good Privacy), PocketScope, Pop-Up, PowerTelnet, Pretty Good Privacy, PrimeSupport, RecoverKey, RecoverKey-International, ReportMagic, RingFence, Router PM, Safe & Sound, SalesMagic, SecureCast, Service Level Manager, ServiceMagic, Site Meter, Sniffer, SniffMaster, SniffNet, Stalker, Statistical Information Retrieval (SIR), SupportMagic, Switch PM, TeleSniffer, TIS, TMach, TMeg, Total Network Security, Total Network Visibility, Total Service Desk, Total Virus Defense, T-POD, Trusted Mach, Trusted Mail, Uninstaller, Virex, Virex-PC, Virus Forum, ViruScan, VirusScan, VShield, WebScan, WebShield, WebSniffer, WebStalker WebWall, and ZAC 2000* are registered trademarks of Network Associates and/or its affiliates in the US and/or other countries. All other registered and unregistered trademarks in this document are the sole property of their respective owners.

The IDEA(tm) cryptographic cipher described in U.S. patent number 5,214,703, licensed from Ascom Tech AG; and the Northern Telecom Ltd., CAST Encryption Algorithm, licensed from Northern Telecom, Ltd. IDEA is a trademark of Ascom Tech AG. Network Associates Inc. may have patents and/or pending patent applications covering subject matter in this software or its documentation; the furnishing of this software or documentation does not give you any license to these patents. The compression code in PGP is by Mark Adler and Jean-Loup Gailly, used with permission from the free Info-ZIP implementation. LDAP software provided courtesy University of Michigan at Ann Arbor, Copyright © 1992-1996 Regents of the University of Michigan. All rights reserved. This product includes software developed by the Apache Group for use in the Apache HTTP server project (<http://www.apache.org/>). Copyright © 1995-2000 The Apache Group. All rights reserved. See text files included with the software or the PGP web site for further information. Balloon help support courtesy of James W. Walker. This software is based in part on the work of the Independent JPEG Group. Soft TEMPEST font courtesy of Ross Anderson and Marcus Kuhn. Biometric word list for fingerprint verification courtesy of Patrick Juola.

Table of Contents

Preface	ix
Before you begin	ix
Organization of this Guide	ix
Conventions used in this Guide	x
How to contact PGP Security and Network Associates	x
Customer service	x
Technical support	xi
Download support	xii
Network Associates training	xii
Comments and feedback	xii
Recommended readings	xiii
The history of cryptography	xiii
Technical aspects of cryptography	xiii
Politics of cryptography	xv
Network security	xvi
Chapter 1. Introducing PGP	17
What can PGP do for you?	17
‘Key’ concepts	17
Basic steps for using PGP	18
Chapter 2. Getting Started	23
Setting up PGP	23
Setting the location of PGP files	23
Starting PGP	25
Making PGP compatible with PGP 2.6.2	26
PGP command syntax	26
Entering configuration parameters on the command line	26
Summary of commands	27
Specifying keys using the user ID	27
Specifying keys using the key ID	28
Cancelling an operation	28

Chapter 3. Creating and Exchanging Keys29

- Choosing a key type29
- Creating a key pair (the -kg option)30
 - Creating subkeys34
- Creating a passphrase that you will remember35
- Working with public and private keyrings36
 - Changing the location or names of your keyrings36
 - Backing up your keys37
 - Protecting your keys37
- Viewing your keys (the -kv option)38
 - On your public keyring38
 - On your private keyring39
 - On other keyrings39
 - Getting more information about keys39
- Exchanging keys with others40
 - Extracting your key to a file (the -kx option)40
 - Adding a key to your keyring (the -ka option)41
 - Removing a key from your keyring (the -kr option)41
 - Adding your key to a key server42
 - Getting someone’s public key from a key server42

Chapter 4. Encrypting and Decrypting Information43

- Exchanging encrypted information43
 - Getting the recipient’s public key43
- Encrypting information43
 - Encrypting with conventional encryption (the -c option)43
 - Encrypting with public key encryption (the -e option)44
 - Encrypting to multiple recipients45
 - Encrypting information to a group46
 - Automatically encrypting to your own key46
 - Encrypting for viewing by recipient only46
- Decrypting information47

Viewing the decrypted file47
Viewing decrypted plaintext output on your screen47
Renaming the decrypted plaintext output file48
Recovering the original plaintext filename48
Chapter 5. Working with Digital Signatures and Validation49
Signing information (the -s option)49
Producing a clear-signed message49
Signing with a specific private key (the -u option)50
Signing and encrypting50
Signing a plaintext ASCII text file51
Creating a detached signature (the -sb option)51
Verifying a digital signature52
Verifying a detached signature52
Storing signed files: signing a file without encrypting52
Validity and trust53
Checking a key's validity53
Granting trust for key validations54
Signing a key (the -ks option)54
Adding an expiration date to your signature (the -ksx option)55
Removing signatures from your key (the -krs option)55
Chapter 6. Key Management Operations57
Managing your keyring57
To view all the keys on a specific keyring57
To remove a key or a user ID from your public keyring58
Editing your key (the -ke option)58
Editing your user ID59
Setting your default user ID60
Editing your passphrase60
Editing trust options for your key61
Adding a designated revoker to your key62
Verifying the contents of your public keyring63
Revoking a key64
Disabling a key65

- Splitting and rejoining a key65
 - Creating a split key65
 - Reconstituting a split key67
 - Reconstituting a split key locally67
 - Reconstituting a split key over the network68
- Additional Decryption Keys69
 - Recover data in an emergency70
 - Data recovery versus key recovery70
 - Types of ADKs70
- Additional Decryption Key policy71
 - Protecting your Additional Decryption Key71
 - Implementing your Additional Decryption Keys71
- Deleting a key from a key server72

Chapter 7. Advanced Topics73

- Using scripts with PGP73
 - Suppressing unnecessary questions74
 - Eliminating confirmation questions74
 - Understanding PGP exit status codes75
 - Using PGP as a UNIX-style filter75
- Working with ASCII and binary data76
 - Encrypting and transmitting binary data76
 - Sending binary data files in ASCII-armored format without encryption or signature77
 - Decrypting ASCII-armored messages77
 - Sending a public key in ASCII-armored format77
 - Sending ASCII text files to different machine environments77
- Wiping your disk78
- Alternative ways to work with passphrases79
 - Storing your passphrase with PGPPASS79
 - Passing your passphrase from another application80
 - PGPPASSFD80
- Working with groups80
 - Creating a group81
 - Add recipients to a group (the -ga option)81

Viewing a group (the -gv and -gvv options)81
Remove recipients from a group (the -gr option)81
Chapter 8. PGP's Configuration File83
Learning about PGP's configuration file83
Specifying configuration values83
Setting configuration parameters from the command line84
Configuration parameters84
Appendix A. Exit and Error Codes105
Appendix B. PGP Command Line Options107
A quick reference of PGP options107
Key options107
Email and file options109
Group options111
Common PGP functions111
Key commands112
Encryption commands114
Decryption commands115
Signing commands116
Group commands117
ASCII and binary data commands117
Help commands118
Appendix C. Attaching a Regular Expression to a Signature119
Attaching a regular expression to a signature119
Definitions of the regular expression syntax used in PGP120
Index121

Preface

PGP is part of your organization's overall security solution for protecting one of your most important assets: *information*. Corporations have traditionally put locks on their doors and require employees to show identification to get into the building. PGP is a valuable tool to help you protect the security and integrity of your organization's data and messages. For many companies, loss of confidentiality means loss of business.

Before you begin

This Guide describes how to use PGP Command Line. It assumes you are familiar with the concepts of public key cryptography as covered in the book *An Introduction to Cryptography* (included with this product).

Organization of this Guide

This Guide is divided into the following sections:

- [Chapter 1, “Introducing PGP,”](#) provides an introduction to using PGP.
- [Chapter 2, “Getting Started,”](#) describes where PGP files are located on your machine, how to start PGP, and PGP command syntax.
- [Chapter 3, “Creating and Exchanging Keys,”](#) describes how to make and exchange keys.
- [Chapter 4, “Encrypting and Decrypting Information,”](#) describes how to encrypt and decrypt your files.
- [Chapter 5, “Working with Digital Signatures and Validation,”](#) defines the concepts of validation and trust in PGP and describes how to create and validate digital signatures.
- [Chapter 6, “Key Management Operations,”](#) describes how to perform administrative tasks on keys and keyrings, including key editing, key splitting, key disabling, key revoking, removing keys from a key server, and creating additional decryption keys (ADKs).
- [Chapter 7, “Advanced Topics,”](#) describes how to use PGP non-interactively from UNIX shell scripts and MSDOS batch files, how to use PGP as a UNIX-style filter, how to encrypt and transmit binary data, how to wipe files, how to work with groups, and alternative ways to work with passphrases.

- [Chapter 8, “PGP’s Configuration File,”](#) defines the parameters of PGP’s configuration file.
- [Appendix A, “Exit and Error Codes,”](#) lists the exit codes and error messages you may encounter while using PGP.
- [Appendix B, “PGP Command Line Options,”](#) is a quick reference to PGP’s options. Includes a syntax guide.
- [Appendix C, “Attaching a Regular Expression to a Signature,”](#) describes the purpose, lists the characters, and defines the syntax for attaching a regular expression to a signature.

Conventions used in this Guide

The following describes the conventions used in this guide:

Bold, sans-serif font	Pathnames, filenames, syntax, and special keys on the keyboard are shown in a bold, sans-serif font.
<i>Italic sans-serif font</i>	Command-line text for which you must supply a value is shown in italic, sans-serif type.
angle brackets < >	Angle brackets (<>) indicate a variable. You supply a value of the type indicated.
square brackets []	Square brackets ([]) indicate an option. The value indicated is not required.

How to contact PGP Security and Network Associates

Customer service

Network Associates continues to market and support the product lines from each of the new independent business units. You may direct all questions, comments, or requests concerning the software you purchased, your registration status, or similar issues to the Network Associates Customer Service department at the following address:

Network Associates Customer Service
4099 McEwen, Suite 500
Dallas, Texas 75244
U.S.A.

The department's hours of operation are 8:00 a.m. to 8:00 p.m. Central time, Monday through Friday.

Other contact information for corporate-licensed customers:

Phone: (972) 308-9960
Email: services_corporate_division@nai.com
Web: <http://support.nai.com/>

Other contact information for retail-licensed customers:

Phone: (972) 308-9960
Email: cust_care@nai.com
Web: <http://www.pgp.com/>

Technical support

PGP Security and Network Associates are famous for their dedication to customer satisfaction. The companies have continued this tradition by making their sites on the World Wide Web valuable resources for answers to technical support issues. PGP Security encourages you to make this your first stop for answers to frequently asked questions, for updates to PGP Security and Network Associates software, and for access to news and virus information.

Web: <http://support.nai.com/>

If the automated services do not have the answers you need, contact Network Associates at one of the following numbers Monday through Friday between 8:00 A.M. and 8:00 P.M. Central time to find out about Network Associates technical support plans.

For corporate-licensed customers:

Phone: (972) 308-9960

For retail-licensed customers:

Phone: (972) 855-7044

To provide the answers you need quickly and efficiently, the Network Associates technical support staff needs some information about your computer and your software. Please include this information in your correspondence:

- Program name and version number
- Computer brand and model
- Any additional hardware or peripherals connected to your computer
- Operating system type and version numbers
- Network name, operating system, and version
- Network card installed, where applicable
- Modem manufacturer, model, and bits-per-second rate, where applicable
- Relevant browsers or applications and their version numbers, where applicable
- How to reproduce your problem: when it occurs, whether you can reproduce it regularly, and under what conditions
- Information needed to contact you by voice, fax, or email

Download support

To get help with navigating or downloading files from the Network Associates Web sites or FTP sites, call:

Corporate customers	(801) 492-2650
Retail customers	(801) 492-2600

Network Associates training

For information about scheduling on-site training for any PGP Security or Network Associates product, call Network Associates Customer Service at: (972) 308-9960.

Comments and feedback

PGP Security appreciates your comments and reserves the right to use any information you supply in any way it believes appropriate without incurring any obligation whatsoever. Please send any documentation comments to tns_documentation@nai.com.

Recommended readings

This section identifies Web sites, books, and periodicals about the history, technical aspects, and politics of cryptography, as well as trusted PGP download sites.

The history of cryptography

- *The Code Book: The Evolution of Secrecy from Ancient Egypt to Quantum Cryptography*, Simon Singh, Doubleday & Company, Inc., September 2000; ISBN: 0385495323. This book is an excellent primer for those wishing to understand how the human need for privacy has manifested itself through cryptography.
- *The Codebreakers: The Story of Secret Writing*, David Kahn, Simon & Schuster Trade, 1996, ISBN 0-684-83130-9 (updated from the 1967 edition). This book is a history of codes and code breakers from the time of the Egyptians to the end of WWII. Kahn first wrote it in the sixties—this is the revised edition. This book won't teach you anything about how cryptography is done, but it has been the inspiration of the whole modern generation of cryptographers.

Technical aspects of cryptography

Web sites

- www.iacr.org—International Association for Cryptologic Research (IACR). The IACR holds cryptographic conferences and publishes journals.
- www.pgpi.org—An international PGP Web site, which is not maintained by PGP Security, Inc. or Network Associates, Inc., is an unofficial yet comprehensive resource for PGP.
- www.nist.gov/aes—The National Institute of Standards and Technology (NIST) Advanced Encryption Standard (AES) Development Effort, perhaps the most interesting project going on in cryptography today.
- www.ietf.org/rfc/rfc2440.txt—The specification for the IETF OpenPGP standard.

Books and periodicals

- *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd edition, Bruce Schneier, John Wiley & Sons, 1996; ISBN 0-471-12845-7. If you can only buy one book to get started in cryptography, this is the one to buy.
- *Handbook of Applied Cryptography*, Alfred Menezes, Paul van Oorschot and Scott Vanstone, CRC Press, 1996; ISBN 0-8493-8523-7. This is the technical book you should get after Schneier. There is a lot of heavy-duty math in this book, but it is nonetheless usable for those who do not understand the math.
- *Journal of Cryptology*, International Association for Cryptologic Research (IACR). See www.iacr.org.
- *Advances in Cryptology*, conference proceedings of the IACR CRYPTO conferences, published yearly by Springer-Verlag. See www.iacr.org.
- *Cryptography for the Internet*, Philip Zimmermann, Scientific American, October 1998 (introductory tutorial article).
- *The Twofish Encryption Algorithm: A 128-Bit Block Cipher*, Bruce Schneier, et al, John Wiley & Sons, Inc., 1999; ISBN: 0471353817. Contains details about the Twofish cipher ranging from design criteria to cryptanalysis of the algorithm.

Politics of cryptography

Web sites

- www.epic.org—Electronic Privacy Information Center.
- www.cripto.org—Internet Privacy Coalition.
- www.eff.org—Electronic Frontier Foundation.
- www.privacy.org—The Privacy Page. Great information resource about privacy issues.
- www.cdt.org—Center for Democracy and Technology.
- www.pgp.com/phil—Phil Zimmermann’s home page, his Senate testimony, and so on.

Books

- *Privacy on the Line: The Politics of Wiretapping and Encryption*, Whitfield Diffie and Susan Landau, The MIT Press, 1998, ISBN 0-262-04167-7. This book is a discussion of the history and policy surrounding cryptography and communications security. It is an excellent read, even for beginners and non-technical people. Includes information that even a lot of experts don’t know.
- *Technology and Privacy: The New Landscape*, Philip Agre and Marc Rotenberg, The MIT Press, 1997; ISBN 0-262-01162-x.
- *Building in Big Brother, The Cryptographic Policy Debate*, edited by Lance Hoffman, Springer-Verlag, 1995; ISBN 0-387-94441-9.
- *The Official PGP User’s Guide*, Philip Zimmermann, The MIT Press, 1995; ISBN 0-262-74017-6. How to use PGP, written in Phil’s own words.
- *The Code Book: The Evolution of Secrecy from Ancient Egypt to Quantum Cryptography*, Simon Singh, Doubleday & Company, Inc., September 2000; ISBN: 0385495323. This book is an excellent primer for those wishing to understand how the human need for privacy has manifested itself through cryptography.

Network security

Books

- *Building Internet Firewalls*, Elizabeth D. Zwicky, D. Brent Chapman, Simon Cooper, and Deborah Russell (Editor), O'Reilly & Associates, Inc., 2000; ISBN: 1565928717. This book is a practical guide to designing, building, and maintaining firewalls.
- *Firewalls and Internet Security: Repelling the Wily Hacker*, William R. Cheswick, Steven M. Bellovin, Addison Wesley Longman, Inc., 1994; ISBN: 0201633574. This book is a practical guide to protecting networks from hacker attacks through the Internet.
- *Hacking Exposed: Network Security Secrets and Solutions*, Stuart McClure, Joel Scambray, and George Kurtz, The McGraw-Hill Companies, 1999; ISBN: 0072121270. The state of the art in breaking into computers and networks, as viewed from the vantage point of the attacker and the defender.

Welcome to PGP Command Line, a member of the PGP product family. PGP products bring easy-to-use, strong encryption and authentication services to your enterprise across a broad range of platforms and applications. With PGP, you can protect your data by encrypting it so that only intended co-workers and business partners can read it. You can also digitally sign data, which ensures its authenticity and that it has not been altered along the way.

What can PGP do for you?

This command line version of PGP is designed to provide several critical security services for UNIX and Windows based workstation users. These services include:

- Secure email communications with co-workers and business partners. You can easily secure email messages using PGP's world-renowned encryption and digital signature capabilities. PGP provides strong encryption services across a broad array of operating systems and email platforms.
- Secure files stored on your file system. You can use PGP to encrypt, decrypt, sign, or verify files stored on your file system. You can either encrypt files to users PGP public keys, or you can use conventional encryption which requires a passphrase to decrypt the file.
- Create and manage PGP keys. You can use PGP to create, view, and maintain your own PGP keypair as well as any public keys of other users that you have added to your public keyring. You can also use PGP key servers to share PGP keys with others.
- Permanently erase files and directories. You can use PGP to thoroughly delete your sensitive files and directories without leaving fragments of their data behind.

'Key' concepts

PGP is based on a widely accepted and highly trusted *public key encryption* system, as shown in [Figure 1-1 on page 18](#), in which two complementary keys, called a *key pair*, are used to maintain secure communications. One of the keys is designated as a *private key* to which only you have access and the other is a *public key* which you freely exchange with other PGP users. Both your private and your public keys are stored in *keyring files*.

As its name implies, only you have access to your private key, but in order to correspond with other PGP users you need a copy of their public keys and they need a copy of yours. You use your private key to sign the email messages and file attachments you send to others and to decrypt the messages and files they send to you. Conversely, you use the public keys of others to send them encrypted email and to verify their digital signatures.

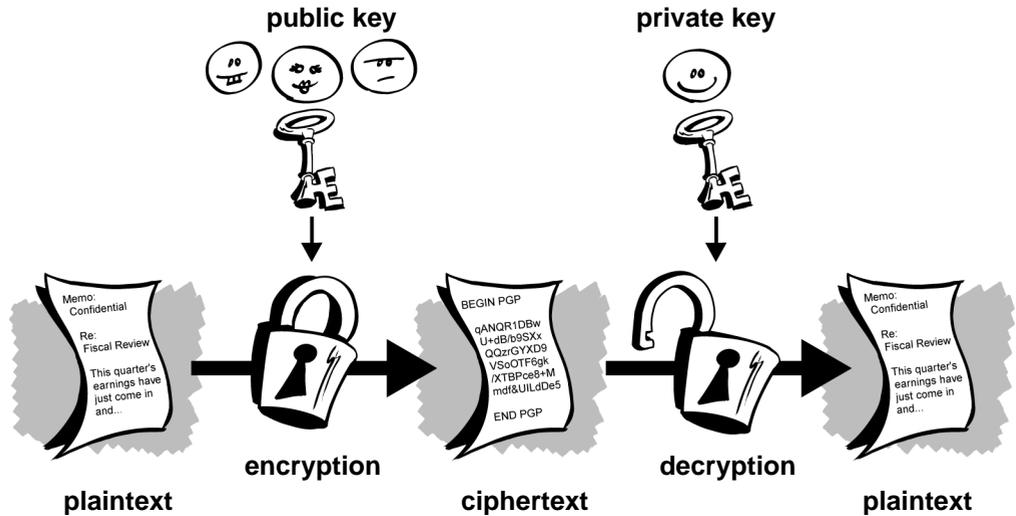


Figure 1-1. Public Key Cryptography diagram

For a comprehensive overview of PGP encryption technology, refer to “*An Introduction to Cryptography*,” which is included with the product.

Basic steps for using PGP

This section takes a quick look at the procedures a user would normally follow in the course of using PGP.

For details concerning any of these procedures, refer to the appropriate chapters in this book.

The order in which you perform the following tasks will vary.

1. Install PGP on your computer.

You will find detailed installation instructions in an accompanying *Installation Guide* or *ReadMe* file.

2. Configure PGP to meet your needs.

You can configure PGP to perform in a specific way. For example, you can specify what encryption and hash algorithms PGP should use, tell PGP to add a specific comment to everything you encrypt, always encrypt a copy of your data to your own key as well as your recipient's, set the level of skepticism PGP should use in determining whether others' keys are valid, and so on.

You do this by setting values in the PGP configuration file, `pgp.cfg`, as described in [Chapter 8, "PGP's Configuration File"](#)

3. Create a private and public key pair.

To use PGP, you need a key pair. A PGP key pair, as described in the section "['Key' concepts](#)," above, is composed of a private key to which only you have access and a public key that you can copy and make freely available to everyone with whom you exchange information.

You can create a new key pair any time after you have finished the PGP installation procedure.

For more information about creating a private and public key pair, refer to "[Creating a key pair \(the -kg option\)](#)" on page 30.

4. Exchange public keys with others (optional).

After you have created a key pair, you can begin corresponding with other PGP users. You will need a copy of their public key and they will need yours. Your public key is just a block of text, so it's quite easy to trade keys with someone. You can include your public key in an email message, copy it to a file, or post it on a public or corporate key server where anyone can get a copy when they need it.

You can, of course, use some of PGP's functionality without exchanging keys with others. However, to encrypt information to another person, you need his or her public key, and conversely, others will need your key if they wish to encrypt information to you.

You can keep copies of others' keys stored in your public keyring file.

For more information about exchanging public keys, refer to and "[Exchanging keys with others](#)" on page 40.

5. Validate public keys.

Validation is a tricky concept within PGP, and you might want to read about it in more detail in the accompanying *Introduction to Cryptography*. Validation goes hand-in-hand with a concept called *trust*. In a nutshell, once you've obtained a copy of someone's public key, you can tell PGP how you feel about the key--whether or not you've made sure that the key has not been tampered with and that it really belongs to the purported owner. You can also tell PGP whether or not and to what degree you *trust* the owner of the key to make such checks on other keys. If you tell PGP you trust the key's owner, then PGP will consider valid any keys the trusted person validates. Persons you trust are called trusted introducers.

To **validate** a key, you compare the unique *fingerprint* on your copy of someone's public key to the fingerprint on that person's original key. If it checks out, you can then *digitally sign* the valid key (using your own private key) to tell others (and PGP) that you consider it valid.

To **trust** a key's owner (not the key, the owner), you set *trust values* in PGP.

Your Corporate Security Officer can act as a trusted introducer, and you may then consider any keys signed by the corporate key to be valid keys. If you work for a large company with several locations, you may have regional introducers, and your Security Officer may be a *meta-introducer*, which is a trusted introducer of trusted introducers.

For more information checking validation and setting trust, see [Chapter 5, "Working with Digital Signatures and Validation."](#)

6. Encrypt and sign your email and files.

After you have generated your key pair and have exchanged public keys, you can begin encrypting and digitally signing email messages and files.

For more information on encryption, see [Chapter 4, "Encrypting and Decrypting Information."](#) For more information on digital signatures, see [Chapter 5, "Working with Digital Signatures and Validation."](#)

7. Decrypt and verify your email and files.

When someone sends you encrypted data, you can decrypt the contents and verify any appended signature to make sure that the data originated with the alleged sender and that it has not been altered.

For more information on decryption, see [Chapter 4, “Encrypting and Decrypting Information.”](#) For more information on verifying digital signatures, see [Chapter 5, “Working with Digital Signatures and Validation.”](#)

8. Wipe files.

When you need to permanently delete a file, you can use the wipe command to ensure that the file is unrecoverable. The file is immediately overwritten so that it cannot be retrieved using disk recovery software.

For more information on wiping files, see [“Wiping your disk” on page 78.](#)

Setting up PGP

This chapter describes where PGP files are located on your machine. It also explains how to start PGP.

Setting the location of PGP files

PGP needs to know where the following files are located:

- **Your keyring files.** PGP stores your key pair in two files: the public portion is stored in **pubring.pkr** and the private portion in **secring.skr**. If you add another user's public key to your keyring, it is stored in the public portion of the keyring. The files are created when you run PGP for the first time.
- **The random number seed file.** PGP uses the data in the random seed file (**randseed.rnd**) when it generates session keys. **randseed.rnd** is created when you run PGP for the first time. (See the book, *An Introduction to Cryptography* for more information on the role of *session keys* when using PGP.)
- **The PGP configuration file.** PGP stores a number of user-defined parameters in the configuration text file **pgp.cfg**. A configuration file enables you to define flags and parameters for PGP, eliminating the need to define these parameters at the command line. **pgp.cfg** is created when you run PGP for the first time.
- **The PGP groups file.** PGP stores any groups you create in the file **pgpgroup.pgr**. Groups are like email distribution lists—you use groups to create a list of recipients for your encrypted information. Encrypting information to the group encrypts the information to every key in the group in one operation. **pgpgroup.pgr** is created when you run PGP for the first time.

You can keep these files in any directory. However, you must indicate to PGP where the files are located. You can specify the path to the keyrings and the randseed file in **pgp.cfg**. You can also specify the path to **pgp.cfg** using the environment variable **PGPPATH**. (For more information on **PGPPATH**, see the section "[PGPPATH](#).”)

PGPPATH

PGPPATH is an environment variable that identifies the location of the PGP configuration file, **pgp.cfg**.

Syntax

```
SET PGPPATH= <PGPpathname>
```

For example:

```
SET PGPPATH=C:\PGP
```

Default file locations

Unix:

The first time you start PGP, the software checks to see if **PGPPATH** is set to a particular pathname.

- If **PGPPATH** is defined, the software looks for the **pgp** configuration file (**pgp.cfg**) in the directory specified by **PGPPATH**. If **pgp.cfg** does not exist in the directory specified, PGP creates it.
- If **PGPPATH** is not defined, the software looks for **pgp.cfg** in the user's home directory, as defined by the environment variable **HOME**. If **pgp.cfg** does not exist, PGP creates the **.pgp** directory within the home directory and creates the **pgp.cfg** file within **.pgp**.

PGP then places the keyring files, the randseed file, and the group file in the **.pgp** directory off your home directory (**HOME/.pgp**) *after* you run PGP for the first time.

Windows NT/2000:

The first time you start PGP, the software checks to see if **PGPPATH** is defined.

- If **PGPPATH** is defined, the software puts the **pgp.cfg** file in the directory specified by **PGPPATH**.
- If **PGPPATH** is not defined, the software checks to see if the environment variable **USERPROFILE** is defined.
 - If **USERPROFILE** is defined, the software puts the **pgp.cfg** file in the **<USERPROFILE>\Application Data\pgp** directory. If **pgp.cfg** does not exist, PGP creates it within this directory.
 - If **USERPROFILE** is not defined, the software puts the **pgp.cfg** file in **<SYSTEMROOT>\pgp**.

PGP then places the keyring files, and the group file in the **<USERPROFILE>\Application Data\pgp** directory *after* you run PGP for the first time.

PGP always places the random number seed file (**randseed.rnd**) in the **<ALLUSERSPROFILE>\Application Data\Network Associates\pgp** directory.

Starting PGP

NOTE: Before you can use **pgp** in Unix, you may need to set the **PATH** variable to point to where **pgp** is installed.

PGP is available at all times. To use PGP, type “**pgp**” and then the command option for the operation you want to perform. You do not need to specifically start or end the program.

pgp *<option>*

The following text appears:

```
PGP(tm) Command Line for Workstation Version #.#
(c) <date> Network Associates, Inc.
Uses the Bsafe(tm) Toolkit, which is copyright RSA Data
Security, Inc.
Export of this software may be restricted by the U.S.
government.
```

For a usage summary, type: **pgp -h**

Making PGP compatible with PGP 2.6.2

This version of PGP includes a compatible switch that enables user-interface compatibility with PGP Version 2.6.2. You may require this feature for incorporation with scripts that parse the output or otherwise interact with PGP dialogs.

To activate this feature, add the following line to the **pgp.cfg** configuration file:

COMPATIBLE=on

You can also enter **pgp +COMPATIBLE** on the command line, although this will make PGP compatible for the duration of the current operation only.

PGP command syntax

You use PGP by typing **pgp** followed by whatever options/parameters you need to perform your operation.

General guidelines:

- Operations involving keys use **-k** options.
- Operations involving groups use **-g** options.
- PGP assumes it is operating on files contained in the current directory. If you want to perform an operation on a file in a different directory, you must specify the file's pathname.

Entering configuration parameters on the command line

Note that any of the PGP configuration parameters described in [Chapter 8, "PGP's Configuration File,"](#) can be entered as long options on the command line.

Type **pgp** followed by a plus sign (+) and then the configuration parameter name.

The following examples show syntax.

pgp +FASTKEYGEN

pgp +PASSTHROUGH

pgp +ADKKEY="0xAB12C34D"

+ used on the command line overrides the default and specifies a new configuration for the *current* operation. This value is transient. **+** does not reset the configuration option.

+ `<parameter_name>` sets the parameter to that parameter's default value--if you want a different value, you must specify it.

For example, the default value for **INTERACTIVE** (which tells PGP to ask for confirms during key adds) is **off**.

Thus the following syntax turns **INTERACTIVE** on.

```
pgp +INTERACTIVE
```

Summary of commands

To display a quick command usage summary of PGP, enter the following at the command line:

```
pgp -h
```

Specifying keys using the user ID

The *user ID* is part of every key. When performing tasks with PGP, you typically identify the key you want to use by specifying the key's user ID or a fragment of the user ID.

When specifying the user ID, keep the following in mind:

- Be as specific as you can because PGP grabs the first matching text from the specified keyring. If you have three keys on your keyring whose user IDs begin with "John," (Dr. John Banner, John Huang, and John Schwartz) using "John" will retrieve Dr. John Banner's key.
- To specify multiple word user IDs, enclose the text in quotes. For example, "**Sophie Luu**"
- PGP is not case-sensitive. "John" and "john" are identical to PGP.

Specifying keys using the key ID

In most cases, you enter a user ID or the fragment of a user ID to select a key. However, you can also use the hexadecimal *key ID* to select a key. To do so, enter the key ID, with a prefix of “**0x**”, instead of the user ID:

```
pgp -kv 0x67F796C2
```

This command instructs PGP to display all keys that have **67F796C2** in their key IDs.

This feature is particularly useful if you have two different keys from the same person, with the same user ID. You can pick the correct key by specifying the specific key ID.

Most command syntax in this guide specifies *<userID>*. *<userID>* and *<keyID>* can be used interchangeably.

Cancelling an operation

To cancel the current operation, press CTRL -C at any prompt.

To cancel a long running operation, press CTRL -C at any time.

This section describes how to generate, view, and manage the public and private key pair that you need to correspond with other PGP users. It also explains how to distribute your public key and obtain the public keys of others so that you can begin exchanging private and authenticated information.

Choosing a key type

PGP provides you with two key types to choose from: Diffie-Hellman/DSS and RSA. Versions of PGP prior to 5.0 used RSA keys exclusively. Versions later than 5.0 introduced the ElGamal variant of Diffie-Hellman technology.

With PGP versions 7.0 and above, the RSA key format has been improved to provide support for features previously available only to Diffie-Hellman/DSS keys: support for Additional Decryption Keys (ADKs), designated revokers, multiple encryption subkeys, and photo ID features. These features are not available to users with RSA keys created prior to Version 7.0, now known as RSA Legacy keys.

Which key type is the right choice for you?

- Choose **Diffie-Hellman/DSS** or **RSA** if you want to take advantage of many PGP key features; including, Additional Decryption Keys (ADKs), designated revokers, multiple encryption subkeys, and photo IDs.
- Choose **RSA** or **RSA Legacy** if you plan to correspond with people who are using RSA keys.
- Choose **RSA Legacy** only if those you communicate with are using older versions of PGP; otherwise choose the new **RSA** key format. (The two versions are not compatible with each other.)

NOTE: The RSA key type is only fully compatible with PGP versions 7.0 and above, and other OpenPGP applications.

If you plan to correspond with people who are still using RSA Legacy keys, you might want to generate an RSA Legacy key pair, which is compatible with older versions of the program.

If you choose to create an RSA key, PGP creates an RSA Legacy key by default. To change the default to the newer RSA format, set the configuration parameter **RSAVER** to equal '4', as described in the section "[RSAVER](#)" on [page 97](#), before you create a key pair.

Creating a key pair (the -kg option)

Unless you have already done so while using another version of PGP, the first thing you need to do before sending or receiving encrypted and signed email is create a new key pair. A key pair consists of two keys: a private key that only you possess and a public key that you freely distribute to those with whom you correspond. You generate a new key pair from the PGP command line.

NOTE: If you are upgrading from an earlier version of PGP, you have probably already generated a private key and have distributed its matching public key to those with whom you correspond. In this case, you don't have to make a new key pair (as described in the next section). Instead, use the **PGPPATH** environment variable to identify the location of your existing keyrings. For more information, see "[Unix:](#)" on [page 24](#).

TIP: It's best to create the fewest number of key pairs possible. You generally need only one key pair. However, if you want one key pair for office use and one for home use, consider the potential disadvantages—if you place both public keys on a public key server, will someone who wants to send you encrypted information know which key to use? Will you remember the passphrases for both keys? It's tempting to create multiple sets of keys, but later you might find yourself wishing you hadn't.

To create a Diffie-Hellman/DSS or RSA key pair:

1. Enter the following at the command line:
pgp -kg
2. Choose a key type—either **DH/DSS** (Diffie-Hellman/DSS) or **RSA**.

NOTE: If you want to create RSA keys using the newer format, you must *first* set the **RSAVER** parameter to equal **4** in the PGP configuration file.

- Enter **1**, the default option, to create a **DH/DSS** key.
- Enter **2**, to create an **RSA** key. Unless you changed the **RSAVER** parameter to equal '4' in the PGP configuration file, PGP creates an RSA Legacy key.

RSA Legacy keys do not support subkeys. Continue to [Step 4](#), if you are generating an RSA Legacy key.

3. If you chose **DH/DSS** or the new **RSA** keys (not RSA Legacy keys), you are asked to choose the type of key you want to create:
 - Enter **1**, the default option, to generate a new signing key (to which you can later add an encryption subkey).
 - Enter **2** to generate an encryption subkey for an existing signing key. (For instructions, see the section “[Creating subkeys](#)” on page 34.)
4. Select the size you want the key to be. A larger key size may take a long time to generate, depending on the speed of the computer you are using.

The key size corresponds to the number of bits used to construct your digital key. A larger key is stronger. However, when you use a larger key, it takes more time to encrypt and decrypt. You need to strike a balance between the convenience of performing PGP functions quickly with a smaller key and the increased level of security provided by a larger key.

Unless you are exchanging extremely sensitive information that is of enough interest that someone would be willing to mount an expensive and time-consuming cryptographic attack in order to read it, you are probably safe using a key composed of 1024 bits.

Choose from one of the following options for the ‘master key’ size, or enter the desired key size in bits:

- For a **DH/DSS** key:
 - Enter **1** to select the maximum key size of 1024 bits.
 - Enter any key size you want between 768 bits and 1024 bits.
 - For a new **RSA** key or an **RSA Legacy** key:
 - Enter **1** to select a key size of 1024 bits.
 - Enter **2** to select a key size of 2048 bits.
 - Enter any key size you want between 768 bits and 2048 bits.
5. Enter the text that will comprise your user ID. PGP prompts you with instructions. It’s not absolutely necessary to enter your real name or even your email address. However, using your real name makes it easier for others to identify you as the owner of your public key. For example:

Robert M. Huang <rmh@xyzcorp.com>

If you do not have an email address, use your phone number or some other unique information that would help ensure that your user ID is unique.

6. Enter the number of days your new key will remain valid. You can enter any number from **0** (forever), the default option, to **10950** days.
7. Enter a passphrase, a string of characters or words you want to use to maintain exclusive access to your private key.

NOTE: Your passphrase should contain multiple words and may include spaces, numbers, and punctuation characters. Choose something that you can remember easily but that others won’t be able to guess. The passphrase is case sensitive, meaning that it distinguishes between uppercase and lowercase letters. The longer your passphrase, and the greater the variety of characters it contains, the more secure it is. Strong passphrases include upper and lowercase letters, numbers, punctuation, and spaces but are more likely to be forgotten. See [“Creating a passphrase that you will remember” on page 35](#), for more information about choosing a passphrase.

8. When prompted, enter the same passphrase again for confirmation.
9. If you chose **DH/DSS** or the new **RSA** keys (not RSA Legacy keys), you are asked if you want to create an encryption subkey at this time.

Type **y** to create an encryption subkey, or type **n** if you do not want to create an encryption subkey. If you do not want to create a subkey at this time, then continue directly to [Step 12](#).

10. If you typed **y** to create an encryption subkey, enter the key size for the new subkey. Choose from one of the following options, or enter the desired key size in bits.
 - For a **DH/DSS** key:
 - Enter **1** to select a key size of 1024 bits.
 - Enter **2** to select a key size of 2048 bits.
 - Enter **3** to select a key size of 3072 bits.
 - Enter any key size you want between 768 bits and 4096 bits.
 - For a new **RSA** key:
 - Enter **1** to select a key size of 1024 bits.
 - Enter **2** to select a key size of 2048 bits.
 - Enter any key size you want between 768 bits and 2048 bits.
11. Enter the number of days your new encryption subkey will remain valid. You can enter any number from **0** (forever), the default option, to **10950** days.
12. If prompted, enter random text to help the PGP software accumulate some random bits to create the keys. Enter keystrokes that are reasonably random in their timing.
13. The generated key pair is placed on your public and private keyrings.
14. Use the **-kv** option to view your new key pair.

```
pgp -kv <userID>
```

15. We recommend that you use the **-kx** command option to copy your new public key from your public keyring and place it in a separate public key file suitable for distribution to your friends. The public key file can be sent to your friends for inclusion in their public keyrings. For more information, see [“Exchanging keys with others”](#) on page 40.

Creating subkeys

NOTE: RSA Legacy keys do not support subkeys.

Every PGP key is actually two keys: a signing key and an encryption subkey. PGP provides the ability to create and revoke new encryption keys without sacrificing your master signing key and the signatures collected on it. One of the most common uses for this feature is to create multiple subkeys that are set to be used during different periods of the key's lifetime.

For example, if you create a key that will expire in three years, you might also create 3 subkeys and use each of them for one of the years in the lifetime of the key. This can be a useful security measure and provides an automatic way to periodically switch to a new encryption key without having to recreate and distribute a new public key.

NOTE: To avoid confusion later, do not overlap your subkeys' validity periods.

To create an encryption subkey:

1. Enter the following at the command line:
pgp -kg
2. Choose the key type. (RSA Legacy keys do not support subkeys.)
3. Enter **2** to generate a new encryption subkey for an existing signing key.
4. Specify the user ID of the signing key to which you want to add the encryption subkey.
5. Type the passphrase for the signing key.
6. Choose a size for the encryption subkey, or enter the desired key size in bits.
 - For a **DH/DSS** key:
 - Enter **1** to select a key size of 1024 bits.
 - Enter **2** to select a key size of 2048 bits.
 - Enter **3** to select a key size of 3072 bits.
 - Enter any key size you want between 768 bits and 4096 bits.

- For a new **RSA** key:
 - Enter **1** to select a key size of 1024 bits.
 - Enter **2** to select a key size of 2048 bits.
 - Enter any key size you want between 768 bits and 2048 bits.
- 7. Type the number of days your new encryption subkey will remain valid. You can enter any number from **0** (forever), the default option, to **10950** days. This sets your subkey's expiration date.
- 8. If prompted, enter random data to use for the key generation process.
- 9. PGP creates the subkey. You can view the subkey using the **-kv** (key view) option as shown below.

```
pgp -kv <the_signing_key's_userID>
```

Creating a passphrase that you will remember

Encrypting a file and then finding yourself unable to decrypt it is a painful lesson in learning how to choose a passphrase you will remember. Most applications require a single word password between three and eight letters. For a couple of reasons we do not recommend that you use a single-word passphrase. A single word password is vulnerable to a dictionary attack, which consists of having a computer try all the words in the dictionary until it finds your password. To protect against this manner of attack, it is widely recommended that you create a word that includes a combination of upper and lowercase alphabetic letters, numbers, punctuation marks, and spaces. This results in a stronger password, but an obscure one that you are unlikely to remember easily.

Trying to thwart a dictionary attack by arbitrarily inserting a lot of funny non-alphabetic characters into your passphrase has the effect of making your passphrase too easy to forget and could lead to a disastrous loss of information because you can't decrypt your own files. A multiple word passphrase is less vulnerable to a dictionary attack. However, unless the passphrase you choose is something that is easily committed to long-term memory, you are unlikely to remember it verbatim. Picking a phrase on the spur of the moment is likely to result in forgetting it entirely. Choose something that is already residing in your long-term memory. It should not be something that you have repeated to others recently, nor a famous quotation, because you want it to be hard for a sophisticated attacker to guess. If it's already deeply embedded in your long-term memory, you probably won't forget it.

Of course, if you are reckless enough to write your passphrase down and tape it to your monitor or to the inside of your desk drawer, it won't matter what you choose.

Working with public and private keyrings

Your PGP keys are stored in two files, called the public and private keyrings:

- **secring.skr** contains the private portion of your key pair. To protect it, PGP stores the key encrypted to your passphrase.
- **pubring.pkr** contains your public key. You can add to the keyring the public keys of everyone with whom you exchange messages.

The keyrings contain binary information, and thus you can't view or manipulate their contents directly.

PGP operates on the keyrings as a pair. If you perform an operation on your public keyring, PGP automatically tries to open the corresponding private keyring.

Changing the location or names of your keyrings

By default, PGP looks for the files **pubring.pkr** and **secring.skr**. If you choose to rename your keyrings, you must specify the keyrings' names in PGP's configuration file (using the **PUBRING** and **SECRING** parameters).

Unix

- The default path for **pubring.pkr** is **<HOME>/pgp/pubring.pkr**
- The default path for **secring.skr** is **<HOME>/pgp/secring.skr**

Windows NT/2000

- The default path for **pubring.pkr** is **<USERPROFILE>\Application Data\pgp\pubring.pkr**
- The default path for **secring.skr** is **<USERPROFILE>\Application Data\pgp\secring.skr**

You can copy your keyring files to another location on your hard drive or to a floppy disk. By default, the keyrings are stored along with the other program files in the directory identified by the **PGPPATH** environment variable, but you can save backups in any location you like. For more information, see ["PGPPATH" on page 24](#).

Backing up your keys

IMPORTANT: PGP Command Line does not automatically back up your keyrings.

Once you have generated a key pair, it is wise to put a copy of it in a safe place in case something happens to the original. Copy your keyring files as you would any other file.

Your private and public keys are stored in separate keyring files. You can copy them to another location on your hard drive or to a floppy disk. You can save your backups in any location you like. For more information on the default keyring locations, see [“Changing the location or names of your keyrings” on page 36](#).

Protecting your keys

Besides making backup copies of your keys, you should be especially careful about where you store your private key. Even though your private key is protected by a passphrase that only you should know, it is possible that someone could discover your passphrase and then use your private key to decipher your email or forge your digital signature. For instance, somebody could look over your shoulder and watch the keystrokes you enter or intercept them on the network.

To prevent anyone who might happen to intercept your passphrase from being able to use your private key, you should store your private key only on your own computer. If your computer is attached to a network, you should also make sure that your files are not automatically included in a system-wide backup where others might gain access to your private key. Given the ease with which computers are accessible over networks, if you are working with extremely sensitive information, you may want to keep your private key on a floppy disk, which you can insert like an old-fashioned key whenever you want to read or sign private information.

As another security precaution, consider assigning a different name to your private keyring file and then storing it somewhere other than in the default PGP folder where it will not be so easy to locate. However, be aware that you need to let PGP know where it is.

Viewing your keys (the **-kv** option)

Viewing your keys is the most basic key management operation. You can list them using variations of the **-kv** option.

NOTE: All operations on your keyrings actually apply to *both* keyrings at once. PGP cannot open just the private keyring or just the public keyring.

An exception to this is the ‘key extract (**-kx**)’ operation--passing in the public keyring name results in an extraction of the public key *only*; passing in the private keyring name extracts both the private and public keys to a file.

TIP: You can use the **WITH-COLONS** and **WITH-DELIMITER** parameters to format the output of your key view. For more information on setting these parameters, see [Chapter 8, “PGP’s Configuration File”](#).

On your public keyring

Type **-kv** to view the keys on your public keyring.

pgp -kv [<userID>] [{<keyring_filename>} or [<keyserver_URL>]}

Examples:

- To display all the keys on your public keyring (**pubring.pkr**), you might enter:

pgp -kv pubring.pkr

- Suppose you want to display Rajesh Sadwahni’s key, which is located on your public keyring.

pgp -kv “Rajesh Sadwahni” pubring.pkr

- Now suppose you want to display Rajesh’s key, which is instead located on your corporate key server.

pgp -kv “Rajesh Sadwahni” ldap://keyserver.mycorp.com

On your private keyring

To view the keys on your private keyring, specify the name of your private keyring:

```
pgp -kv [<userID>] [{<keyring_filename>} or [<keyserver_URL>]}
```

For example:

```
pgp -kv "Jordan" secring.skr
```

On other keyrings

To view keys on another keyring, specify the keyring's name.

Keep in mind that PGP will try to open both the public and private keyrings at once; the private portion of the keyring must be in the same directory as the public keyring and must either have the filename **secring.skr**, or you must specify its new name using the **SECRING** configuration parameter. You can set the **SECRING** parameter's value either in the configuration file or on the command line as shown below.

```
pgp +SECRING=<path> / <secret_key_filename>
```

For example:

```
pgp +SECRING=<HOME>/pgp/secring.skr
```

Getting more information about keys

To see additional information on your key, for example, any signatures associated with the key, use the **-kvv** (view key and signatures) option.

```
pgp -kvv [<userID>] [<keyring_file>]
```

To verify a signature on your key (that is, confirm that the key is valid), use the **-kc** (key check) option.

```
pgp -kc [<userID>] [<keyring_file>]
```

To see the fingerprint of the key, use the **-kvc** (key view and check) option.

```
pgp -kvc [<userID>] [<keyring_file>]
```

Exchanging keys with others

After you create your keys, you need to make them available to others so that they can send you encrypted information and verify your digital signature. You have three alternatives for distributing your public key:

- Export your public key to a text file.
- Make your public key available through a public key server.
- Include your public key in an email message.

Your public key is basically composed of a block of text, so it is quite easy to make it available through a public key server, include it in an email message, or export or copy it to a file. The recipient can then use whatever method is most convenient to add your public key to their public keyring.

Extracting your key to a file (the **-kx** option)

To export your key to a file, which you can then freely distribute to others or add to a key server, you can use the **-kx** (key extract) option.

This exports only the public portion of your key. (Square brackets ([]) indicate an optional parameter, and angle brackets (< >) indicate a variable in which you supply a value of the type indicated.)

```
pgp -kx <userID> <key_filename> [<keyring_name>]
```

For example:

```
pgp -kx "John Lee" johnkey.pgp pubring.pkr
```

You can then give the file **"johnkey.pgp"** to anyone who wants a copy of your key. (If you plan to email the key to others, use the **-kxa** option as described below.)

To extract the private portion as well, specify the private keyring instead of the public keyring. This extracts *both* parts of the key pair.

For example:

```
pgp -kx "John Lee" johnkey.pgp secring.skr
```

The **-kx** option produces a file with a single, binary key on it.

To extract the key in ASCII-armored format, which makes it easy to paste into email, use the **-kxa** option.

```
pgp -kxa <userID> <key_filename> [<keyring_name>]
```

For example:

```
pgp -kxa "John Lee" johnkey.pgp pubring.pkr
```

In this example, PGP creates a file called `johnkey.pgp.asc`, which you can paste into email.

Adding a key to your keyring (the `-ka` option)

You can add a key to a specified keyring by including the name of the keyring on the command line following the name of the file containing the key you want to add.

```
pgp -ka <key_filename> [<keyring_name>]
```

For example:

```
pgp -ka bobkey.pgp pubring.pkr
```

PGP does not allow you to add duplicate keys to your keyring. If the second key has any differences, such as an additional user ID, PGP merges the changes.

Removing a key from your keyring (the `-kr` option)

To remove someone's public key from your public keyring, use the `-kr` (key remove) option.

```
pgp -kr <userID_of_key_to_remove> [<keyring_name>]
```

For example, suppose you want to remove Gina Marala's key from your public keyring.

```
pgp -kr "Gina Marala" pubring.pkr
```

If you specify a particular keyring file, PGP tries to open that file and the corresponding public or private keyring file. If the key that you want to delete is part of your key pair, PGP asks you if you want to delete the private key as well. If you answer **No**, PGP does not delete anything. If you say **Yes**, PGP deletes the private portion from the private keyring as well as the public portion from the public keyring.

For example, suppose you want to remove a test *key pair* from your keyring.

```
pgp -kr "Test 1" secring.skr
```

PGP prompts you to confirm you want to remove the private key. To remove it, answer **Yes**.

Adding your key to a key server

You can add your key to a key server so that it is available to others. To add your key to a server, you use the **-kx** (key extract) option. Instead of specifying the name of a file to which PGP should extract the key, you specify a key server's URL. PGP then extracts the key from the keyring and places it on the server.

```
pgp -kx <userID> <keyserver_URL>
```

For example:

```
pgp -kx "John Lee" ldap://certserver.pgp.com
```

Getting someone's public key from a key server

Getting someone's key from a key server and putting it on your public keyring is a two-step operation. First you extract the key from the key server by specifying the server's URL, then you add the resulting file to your public keyring.

```
pgp -kx <userID> <key_filename> <keyserver_URL>
```

```
pgp -ka <key_filename> [<keyring_name>]
```

For example:

```
pgp -kx "John Lee" johnkey ldap://certserver.pgp.com
```

The above command extracts the key with the user ID **John Lee** from the key server **certserver.pgp.com** and puts it in a file called **johnkey.pgp**. The following command adds the key to **pubring.pkr**.

```
pgp -ka johnkey.pgp pubring.pkr
```

Exchanging encrypted information

This chapter describes the various PGP options that let you encrypt and decrypt your data.

For an overview of encryption and decryption and a description of how PGP performs the two operations, see *An Introduction to Cryptography*.

Getting the recipient's public key

Before you encrypt, you need to be sure that you are encrypting it with the correct public key. This means you need to check the public key to ensure that it truly belongs to the person to whom you think it belongs. Encrypting the message with the wrong key basically makes it:

- Closed to your intended recipient, and
- Open to whomever's key you encrypted it to (possibly an interloper)

Verifying that a key belongs to its purported owner is discussed in the section [“Validity and trust” on page 53](#).

Encrypting information

Encryption is one of the most common operations you will perform with PGP.

Encrypting with conventional encryption (the `-c` option)

Encrypting with *conventional* encryption means encrypting to a particular passphrase instead of to a public key. Conventional encryption is useful in certain situations, like when you're encrypting to yourself; however, the typical problem one encounters with conventional encryption is the difficulty in securely communicating the passphrase to the recipient.

For more information on conventional encryption, see *An Introduction to Cryptography*.

```
pgp -c <plaintext_filename>
```

To specify a passphrase as well, use the **-z** (passphrase) option. If the passphrase contains spaces, you must enclose the entire string in quotes.

```
pgp -c <plaintext_filename> [-z <passphrase>]
```

The following command encrypts the file **secretdocument.txt** using the passphrase **quick, get a mango**. To decrypt the file, the recipient will have to type in the same passphrase.

```
pgp -c secretdocument.txt -z "quick, get a mango"
```

This results in an encrypted file named **secretdocument.txt.pgp**.

NOTE: Exercise caution when using the **-z** option. Whenever you enter your passphrase as cleartext (as in the example above), you risk its interception.

Encrypting with public key encryption (the **-e** option)

To encrypt information, you use the **-e** (encrypt) option. (The key to which you want to encrypt must be on your keyring.)

NOTE: It is recommended that you reference the key ID instead of the user ID during encryption if you have very similar user IDs or subkeys on your keyring. For example, if you want to encrypt to user ID "smith@nai.com" and you also have a user ID of "jsmith@nai.com" on your keyring, then PGP will encrypt to both users. To ensure that you only encrypt to the intended recipient, reference the key ID of the key to which you want to encrypt.

```
pgp -e <plaintext_filename> <recipient's_userID>
```

For example, to encrypt the file **testresults.doc** to Jennifer Quino's key, you would use the following syntax:

```
pgp -e testresults.doc "Jennifer Quino"
```

Typically, you use the **-e** option in conjunction with other options; the **-ea** (encrypt ASCII) option encrypts information into ASCII text, which is suitable for sending through email channels. The **-t** (encrypt text) option tells PGP that you are encrypting a text file and preserves its text format.

NOTE: Do not use **-t** with binary data, such as a spreadsheet or word processing file.

The following example would put the file **testresults.doc** in a format appropriate for sending via email:

```
pgp -ea testresults.doc "Jennifer Quino"
```

This results in an encrypted file called **testresults.doc.asc**.

If the file **testresults.txt** were a text file, you would use the following option instead:

```
pgp -eat testresults.txt "Jennifer Quino"
```

This results in an encrypted file named **testresults.txt.asc**.

The **-f** (filter) option causes PGP to write its output to *standard output* instead of to a file. You can thus write your encrypted information directly to an application that reads *standard input*.

The following example would encrypt the file **testresults.doc** to the key belonging to Jennifer Quino and filter the encrypted file to an application that reads *standard input*. The **-ef** (encrypt and filter) option only works if the input comes from *standard input*.

```
pgp
```

```
pgp -ef testresults.doc "Jennifer Quino" < testresults.doc >  
testresults.doc.pgp
```

NOTE: For the above example, the angle brackets (**<** **>**) represent the direction of the *standard input* and *standard output*. They do not represent a variable.

Encrypting to multiple recipients

To encrypt to several recipients at once, you can manually specify their public key user IDs as shown in the following syntax.

```
pgp -e <filename> <userID1> <userID2> <userID3>...
```

For example, suppose you want to encrypt meeting minutes to three coworkers in a format you can send via email.

```
pgp -eat mtgminutes.txt "Carol Wong" "Angie Vicari" "Kevin Sprole"
```

You can also create a *group*, which functions much like a mailing or distribution list functions in most email programs. For information on working with groups, see the section, "[Working with groups](#)" on page 80.

Encrypting information to a group

To encrypt information to a predefined group of recipients, you specify the group name as you would a single recipient's name.

```
pgp -e <plaintext_filename> <groupname>
```

For information on managing groups, see “Working with groups” on page 80.

Automatically encrypting to your own key

The configuration parameter **ENCRYPTTOSELF** enables you to automatically encrypt everything to your own key or some other predefined key in addition to any specified recipients.

To set this up, you must set parameters in PGP's configuration file, **pgp.cfg**:

- Set the **MYNAME** parameter to the userID of the desired key.
- Set **ENCRYPTTOSELF** to **on** (or type **+ENCRYPTTOSELF=on** at the command line).

Encrypting for viewing by recipient only

To specify that the recipient's decrypted plaintext be shown only on the recipient's screen and not saved to disk, add the **-m** option:

```
pgp -sem <message.txt> <recipient's_userID>
```

When the recipient decrypts the ciphertext with their secret key and passphrase, the plaintext is displayed on the recipient's screen but is not saved to disk. The text is displayed as it would if the recipient used the UNIX “more” command, one screen at a time. If the recipient wants to read the message again, he or she must decrypt the ciphertext a second time.

This feature is the safest way for you to prevent your sensitive message from being inadvertently left on the recipient's disk.

NOTE: This feature does not prevent a clever and determined person from finding a way to save the decrypted plaintext to disk—it is designed to help prevent a casual user from doing it inadvertently.

Decrypting information

You decrypt information using the private portion of your key pair (unless you've encrypted using conventional encryption, in which case you decrypt using the correct passphrase). You can decrypt only that information which is encrypted to the corresponding public portion of your key pair.

Decrypting with PGP is a matter of using the encrypted file's name as an argument to the PGP command, as shown in the following syntax.

```
pgp < ciphertext_filename >
```

You are required to enter a passphrase for your private key.

You can also specify your passphrase as part of the operation:

```
pgp < ciphertext_filename > [-z < passphrase >]
```

```
pgp secretdocument.asc -z "quick, get a mango"
```

Viewing the decrypted file

When PGP encrypts a plaintext file, it saves the original filename and attaches it to the plaintext before it is compressed and encrypted. When PGP decrypts the ciphertext file, it names the plaintext output file with a name similar to the input ciphertext filename, but drops the extension.

You can specify other output results for the decrypted information as described below.

Viewing decrypted plaintext output on your screen

To view decrypted plaintext output on your screen (similar to the UNIX-style "more" command), without writing the output to a file, use the **-m** (more) option when you decrypt:

```
pgp -m < ciphertext_filename >
```

This command instructs PGP to display the decrypted plaintext on your screen, one screen at a time.

Renaming the decrypted plaintext output file

When PGP decrypts a ciphertext file, it names the plaintext output file with a name similar to the input ciphertext filename, but drops the extension.

Use the **-o** option on the command line to specify a more meaningful plaintext filename for the output:

```
pgp -o <new_plaintext_filename> <original_ciphertext_filename>
```

Recovering the original plaintext filename

As stated in the previous section, when PGP encrypts a plaintext file, it saves the original filename and attaches it to the plaintext before it is compressed and encrypted. Use the **-p** option to instruct PGP to preserve the original plaintext filename and use it as the name of the decrypted plaintext output file.

```
pgp -p <ciphertext_filename>
```

Working with Digital Signatures and Validation

5

For an overview of digital signatures, validation, trust, and the other concepts in this chapter, as well as a description of how PGP performs such tasks, see *An Introduction to Cryptography*.

Signing information (the **-s** option)

To sign a plaintext file using your default private key, use the **-s** (sign) option. If you do not specify another key (using the **-u** option), PGP uses your default key. (Your default key is specified using the **MYNAME** parameter in the PGP configuration file.)

```
pgp -s <plaintext_filename>
```

You must supply the passphrase for the private key.

Unencrypted PGP signed messages have a signature certificate prepended in binary form. The signed message is compressed, rendering the message unreadable to human eyes, even though the message is not encrypted. The following is an example of an unencrypted signed PGP message:

```
-----BEGIN PGP MESSAGE-----  
Version: PGP 7.0.1  
  
owHrZLBnZmWwLJntk/hadk01T+xqQSahWwzzY67c+23aMIvPrqNLedIezbfJ  
DPNr2H8dcjW5FPnMeKXn4+063rt2JpqvZZRLYilJLS6RYGBgCm1IVSgszUzO  
yi/PU0jLr1DIKs0tSE1RyC9LLVioAcrnJFZVKqTkp+txjQzVAAkQKf  
  
-----END PGP MESSAGE-----
```

Producing a clear-signed message

To produce a clear-signed message, one that can be read with human eyes, and without the aid of PGP, the **CLEARSIG** parameter must be set to **on** (the default) in the PGP configuration file, and it must be used in conjunction with the **ARMOR** and **TEXTMODE** parameters. Set **ARMOR=ON** (or use the **-a** option), and set **TEXTMODE=ON** (or use the **-t** option).

For example, you would enter the following on the command line (assuming that **CLEARSIG=on** in the configuration file):

```
pgp -sat <plaintext_filename>
```

The following is an example of a clear-signed message:

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

The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.

-----BEGIN PGP SIGNATURE-----
Version: PGP 7.0.1

owHrZLBnZmWwLJntk/hadk01T+xqQSahWwzzY67c+23aMivPrqNLedIezbfJ
DPNr2H8dcjW5FPnM
=vZZRL

-----END PGP SIGNATURE-----
```

Note that the recipient must still use PGP to verify the signature. For more information on verifying signatures, see [“Verifying a digital signature” on page 52](#). For more information on using the **CLEARSIG** parameter, see [“CLEARSIG” on page 87](#).

Signing with a specific private key (the **-u** option)

If you have more than one private key on your private keyring, PGP automatically uses the default key (specified using the **MYNAME** parameter in **pgp.cfg**) to sign your messages. To sign using a private key that is not your default private key, you must specify a different key using the **-u** (user) option.

```
pgp -s <textfile> -u <userID>
```

You must supply the passphrase for the private key.

Signing and encrypting

To sign a plaintext file with your secret key and encrypt it with the recipient's public key in a single operation, you combine the **-e** (encrypt) option with the **-s** (sign) option. You can optionally specify which private key to use to sign the file.

```
pgp -es <plaintext filename> <recipient's_userID> [-u <your_userID>]
```

Signing a plaintext ASCII text file

To sign a plaintext ASCII text file with your secret key, producing a signed plaintext message suitable for distribution through channels such as email, use the **-t** (text) option. To encrypt and sign a plaintext ASCII text file, producing a message suitable for sending through email, use the following syntax:

```
pgp -est <plaintext filename> <recipient's_userid> [-u <your_userid>]
```

For example, if Cee Wong wants to encrypt **secretfile.txt** to Sean Adams and sign it with her private key, she would enter the following:

```
pgp -est secretfile.txt "sean adams" -u cwong
```

The encrypted and signed file can then be sent through email. The following is an example of an encrypted and signed message:

```
-----BEGIN PGP MESSAGE-----
Version: PGP 7.0.1

aMIvPrqNLedIezbfJDPNr2H8dcjW5FPnMeKXn4+063rt2JpqvZZRLYi1JLS6
RYGBgCm1IVSgszUzOyi/PU0jLr1DIKs0tSE1RyC9LLVioAcrnJFZVKqTkp+t
xjQzVAAkQKfowHrZLBnZmWwLJntk/hadk01T+xqQSahWwowHrZLBnZmWwLJn
tk/hadk01T+xqQSahWwzzY67c+23aMIvPrqNLedIezbfJDPNr2H8dcjW5FPn
MeKXn4+063rt2JpqvZZRLYi1JLS6RYGBgCm1IVSgszUzOyi/PU0jLr1DIKs0
tSE1RyC9LLVioAcrnJFZVKqTkp+txjQzVAAkQKfowHrZLBnZmWwLJntk/had
k01T+xqQSahWw+xqQSahWwowHrZLBnZmWwLJntk/hadk01T+xqQSahWwzzY6
7c+23aMIvPrqNLedIezbfJDPNr2H8dcjW5FPnMeKXn4+063rt2JpqvZZRLYi
1JLS6RYGBgCm1IVSgszUzOyi/PU0jLr1DIKs0tSE1RyC9LLVI==
=kggl

-----END PGP MESSAGE-----
```

Creating a detached signature (the **-sb** option)

In most cases, signature certificates are physically attached to the text they sign. This makes it convenient to verify signatures. You can, however, create a separate, detached signature, and then send both files (the text file and the signature certificate file) to the recipient. This feature is useful when more than one party must sign a document such as a legal contract, without nesting signatures. Each person's signature is independent.

To create a separate, detached signature certificate file, combine the **-b** (break) option with the **-s** (sign) option. You can optionally specify which private key to use to sign the file.

```
pgp -sb <plaintext_filename> [-u <your_userid>]
```

For example:

```
pgp -sb letter.txt
```

This instructs PGP to produce a separate, detached signature certificate in a file named **letter.txt.sig**. The contents of **letter.txt.sig** are not appended to **letter.txt**.

Verifying a digital signature

To determine whether an attached digital signature is valid, you *verify* it. PGP automatically verifies signatures as part of the decryption operation. If you want to verify a file, use the same syntax as that for decryption:

```
pgp <filename>
```

Verifying a detached signature

When you attempt to process a detached signature certificate file, PGP asks you to identify the corresponding text file. Once the text file is identified, PGP checks the signature integrity.

If you know that a signature is detached from a text file, you can specify both filenames on the command line:

```
pgp <signature_filename.sig> <textfile.txt>
```

For example:

```
pgp letter.txt.sig letter.txt
```

If the text file exists in the same directory as the detached signature certificate file, you can enter the following shortened command:

```
pgp letter.txt.sig
```

PGP assumes that the signed text has the same name as the signature (**.sig**) file—if it does not, then you must specify the filename.

Storing signed files: signing a file without encrypting

If you sign a plaintext file without specifying encryption, PGP compresses the file after you sign it. This makes the file unreadable to the casual human observer. This is a suitable way to store signed files in archival applications because it saves space. However, it is not an especially secure means for storing the data.

Validity and trust

Every user in a public key system is vulnerable to mistaking a phony key (certificate) for a real one. *Validity* is confidence that a public key certificate belongs to its purported owner. Validity is essential in a public key environment where you must constantly establish whether or not a particular certificate is authentic.

When you've assured yourself that a key belonging to someone else is valid, you can sign the copy on your keyring to attest to the fact that you've checked the key and that it's an authentic one. If you want others to know that you gave the key your stamp of approval, you can export the signature to a certificate server so that others can see it.

Checking a key's validity

Viewing signatures on a key (the **-kc** and **-kvv** options)

To view the signatures on a key use the **-kc** (key check) option. **-kc** displays all the keys on your keyrings and then, for each key, the signatures on the key. It also displays the level of trust you have in each key and its validity, and verifies the signatures.

```
pgp -kc [<userID>] [<keyring_filename>]
```

To view the contents of the keyring without showing the levels of trust or validity, use the **-kvv** option.

```
pgp -kvv [<userID>] [<keyring_filename>]
```

Viewing a key's fingerprint (the **-kvc** option)

You can check that a certificate is valid by calling the key's owner (so that you originate the transaction) and asking the owner to read his or her key's fingerprint to you and verifying that fingerprint against the one you believe to be the real one.

To do so, both you and the key's owner use the **-kvc** (key view check) command to view the key's fingerprint:

```
pgp -kvc <userID> [<keyring_filename>]
```

This command instructs PGP to display the key with the 40 character digest of the public key components (RSA Legacy keys have 32 character fingerprints). Read the fingerprint to the key's owner to see if the fingerprints match.

Using this procedure, you can verify and sign each other's keys with confidence. This is a safe and convenient way to get the key trust network started for your circle of friends.

Note that sending a key fingerprint via email is not the best way to verify the key because email can be intercepted and modified. It is best to use a different channel than the one that was used to send the key itself. A good combination is to send the key via email, and verify the key fingerprint via a voice telephone conversation. Some people even distribute their key fingerprint on their business cards.

Granting trust for key validations

Trust is confidence in another person's ability to validate a key. If you designate someone a *trusted introducer*, then all keys validated by the trusted introducer are considered to be valid to you.

This means that if you ever get a key from someone that has been signed by an individual whom you have designated as trustworthy, the key is considered valid even though you have not done the check yourself.

Changing your trust settings on a key (the `-ke` option)

To edit the trust parameters for a particular key on your keyring (that is, to designate someone a trusted introducer), you use the `-ke` (key edit) option.

```
pgp -ke <userID> [<keyring_filename>]
```

For more information on editing your key and key management, see [“Editing your key \(the `-ke` option\)”](#) on page 58.

Signing a key (the `-ks` option)

To sign and validate someone else's public key on your public keyring, use the `-ks` (key sign) option. When you sign a key it is automatically considered valid to you.

```
pgp -ks <recipient's_userID> [-u <your_userID>] [<keyring_filename>]
```

For example,

```
pgp -ks "john rice" -u "tim ryans" pubring.pkr
```

NOTE: Be absolutely certain that the key belongs to its purported owner before you sign it!

Adding an expiration date to your signature (the **-ksx** option)

Use the **-ksx** (key sign expiration) option to append an expiration date to your signature on the recipient's key. Set *<NumDays>* equal to the number of days until your signature expires. To set the signature to never expire, set *<NumDays>* equal to zero. To add an expiration date to your signature, use the following syntax:

```
pgp -ksx <NumDays> <recipient's_userID> [-u <your_userID>]
[<keyring_filename>]
```

For example, if Lathisha Jones signs the public key belonging to Gilbert Sampson and wants her signature to expire in two weeks (or 14 days), she would use the following syntax:

```
pgp -ksx 14 "gilbert sampson" -u "latisha jones" pubring.pkr
```

If Lathisha did not want her signature to expire, then she would use the following syntax:

```
pgp -ksx 0 "gilbert sampson" -u "latisha jones" pubring.pkr
```

Removing signatures from your key (the **-krs** option)

Use the **-krs** (key remove signature) option to remove signatures from a local copy of your key. Bear in mind, however, that if others have signed a copy of your key that is residing on a public key server, the signatures will reappear on your key when you synchronize your key with the one on the key server.

To remove selected signatures from a user ID on a key:

```
pgp -krs <userID> [<keyring_filename>]
```

For example,

```
pgp -krs "leah roberts"
```

In the above example, PGP is instructed to remove signatures from the key belonging to Leah Roberts.

Key management is the secure administration of keys or keyrings. Administrative tasks you might perform on your keys include:

- Editing your key--adding or changing user IDs, adding and deleting signatures, changing your passphrase, changing your trust parameters.
- Disabling your key.
- Revoking your key.
- Splitting and rejoining your key.
- Creating additional decryption keys (ADKs).
- Deleting your key from a key server.

Managing your keyring

You may accumulate many keys in the course of using PGP. Over time, you may want to view, update, or remove the keys on the keyring.

To view all the keys on a specific keyring

To see the keys on your keyrings, use the **-kv** (key view) option. You can, optionally, specify the name of the keyring. By default, PGP tries to open both the public and private keyrings at once.

pgp -kv <keyring_filename>

PGP lists the keys in alphabetical order, sorted using the first letter of the key's user ID. The following example shows the contents of a keyring with its contents labeled:

type	key size	key ID	creation date	user ID(s)
RSA	1024	0x7D75EB0F	2000/3/20	Albert Reilly <arielly@domain.com
DSS	2048/1024	0xF5ED0CB	1999/01/22	Corporate Key <CIO@domain.com>
DSS	2048/1024	0xFAEBD5FC	1997/04/07	Philip R. Zimmermann <prz@pgp.com> Philip R. Zimmermann <prz@acm.org>

TIP: You may wish to use a pager (such as **more**) if you have many keys on your keyring.

pgp -kv <keyring_filename> | **more**

To remove a key or a user ID from your public keyring

To remove a key or user ID from your keyring, use the **-kr** (key remove) option.

pgp -kr <userID> [<keyring_filename>]

If the user ID of the key you want to remove is similar to a user ID of another key on your keyring, then reference the key ID of the key you want to remove instead.

pgp -kr <keyID> [<keyring_filename>]

Editing your key (the **-ke** option)

When you think of key management, you probably think of maintenance, such as updates or changes to your key.

For example, you may need to change your passphrase, perhaps because someone looked over your shoulder while you typed it on the keyboard. You may need to change your user ID, because you changed your name or your email address. You may need to add a second or third user ID to your key, because you are known by more than one name, email address, or job title.

You may also need to make an existing key your default signing key. In order to do so, you must set the **MYNAME** parameter in your PGP configuration file, **pgp.cfg**. For more information, see [“MYNAME” on page 93](#).

You can use the **-ke** (key edit) option to perform any of the following actions:

- Edit your user ID.
- Edit your passphrase.
- Change the trust setting on a key.
- Add a designated revoker to your key.

PGP prompts you for your new settings as described in the following sections.

Editing your user ID

If you edit your user ID, PGP actually adds a new user ID, without deleting the old one. If you want to delete an old user ID, you must do that in a separate operation. For more information on deleting a user ID, see [“To remove a key or a user ID from your public keyring” on page 58](#).

Note the following:

- The optional [*<keyring_filename>*] parameter, if specified, must be a public keyring, not a secret keyring.
- The user ID field must be your own user ID, which PGP knows is yours because it appears on both your public keyring and your secret keyring.
- Both keyrings are updated, even though you only specified the public keyring.

To edit your user ID:

1. Enter the following on the command line:

```
pgp -ke <your_userID> [<keyring_filename>]
```

2. Enter your current passphrase to gain access to the key.

PGP asks, “Do you want to add a new user ID?”

3. Type **y** to add a new user ID to this key.

4. Enter the new user ID.

PGP asks, “Change primary user ID?”

5. Enter **y** if you want to make the new user ID your primary user ID, or type **n** if you do not want to change your primary user ID at this time.

Setting your default user ID

Occasionally, you may need to change your default user ID; perhaps because you changed your name or your email address.

To change your default user ID:

1. Enter the following on the command line:

```
pgp -ke <your_userID> [<keyring_filename>]
```
2. Enter your current passphrase to gain access to the key.
PGP asks, “Change primary user ID?”
3. Type **y** to change your primary user ID for this key.
PGP lists each of your user IDs—one at a time—and, for each user ID, asks whether you want to make it your primary user ID.
4. When PGP lists the user ID you want as your primary user ID, type **y**. Type **n** for each of the other user IDs.

Editing your passphrase

Your security is only as good as your passphrase. If you feel that your passphrase has been compromised, then you should change your passphrase immediately.

To change your passphrase:

1. Enter the following on the command line:

```
pgp -ke <your_userID> [<keyring_filename>]
```
2. Enter your current passphrase to gain access to the key.
PGP asks, “Do you want to change your passphrase?”
3. Type **y** to change your passphrase.
4. Enter a passphrase, a string of characters or words you want to use to maintain exclusive access to your private key.
5. When prompted, enter your new passphrase again for confirmation.

Editing trust options for your key

Use the **-ke** option to edit trust options for a key on your keyring. You can turn off “implicit trust” for your own key pair, or you can edit a public key on your keyring, designating someone as a trusted introducer.

If you designate someone a *trusted introducer*, then all keys validated by the trusted introducer are considered to be valid to you.

This means that if you ever get a key from someone that has been signed by an individual whom you have designated as trustworthy, the key is considered valid even though you have not done the check yourself.

To edit trust options for your key pair:

1. Enter the following on the command line:

```
pgp -ke <your_userID> [<keyring_filename>]
```
2. Enter your current passphrase to gain access to the key.
PGP asks, “Do you implicitly trust this key?”
3. Type **y** if you implicitly trust this key, or type **n** if you no longer implicitly trust this key.
If you type **n**, the key becomes untrusted and invalid.

To edit trust options for a public key on your keyring:

1. Enter the following on the command line:

```
pgp -ke <userID_for_public_key> [<keyring_filename>]
```

If you have not already signed this key (or trusted it previously), then PGP displays the current trust for this key’s owner as “untrusted”.

PGP asks, “Would you trust <keyID> to act as an introducer and certify other people’s public keys to you?”

2. Enter one of the following four responses:
 - Enter **1**, the default, if you do not know if you trust the owner of this key to act as a trusted introducer.
 - Enter **2**, if you do not trust the owner of this key to act as a trusted introducer.
 - Enter **3**, if you usually trust the owner of this key to act as a trusted introducer.
 - Enter **4**, if you always trust the owner of this key to act as a trusted introducer.

Adding a designated revoker to your key

It is possible that you might forget your passphrase someday or lose your private key. If this happens, then you would be unable to use your key again, and you would have no way of revoking it to show others not to encrypt to it. To safeguard against this possibility, you can appoint a third-party key revoker. The third-party you designate is then able to revoke your key just as if you had revoked it yourself. For more information on revoking keys, see [“Revoking a key” on page 64](#).

NOTE: For a key to appear revoked to another user, both the revoked key and the Designated Revoker key must be on his/her keyring. Thus, the designated revoker feature is most effective in a corporate setting, where all users' keyrings contain the company's Designated Revoker key. If the revoker's key is not present on a person's keyring, then the revoked key does not appear revoked to that user and he/she may continue to encrypt to it.

NOTE: This feature is available for Diffie-Hellman/DSS and RSA keys. Designated revokers are not supported by RSA Legacy keys.

To add a designated revoker to your key:

1. Ensure that the designated revoker's key is on your keyring.
2. Enter the following on the command line:

```
pgp -ke <your_userID> [<keyring_filename>]
```

3. Enter your current passphrase to gain access to the key.
PGP asks, “Set designated revoker?”
4. Type **y** to add a designated revoker to your key.
5. You will be asked if you are sure you want to add the specified designated revoker to your key. Type **y** if you want to add the specified designated revoker to your key.

You are prompted to enter the user ID of the person who will act as your designated revoker.

Verifying the contents of your public keyring

PGP automatically checks any new keys or signatures on your public keyring and updates all the trust parameters and validity scores. In theory, it keeps all the key validity status information up-to-date as material is added to or deleted from your public keyring.

At some point, however, you may want to explicitly force PGP to perform a comprehensive analysis of your public keyring, checking all the certifying signatures, checking the trust parameters, updating all the validity scores, and checking your own ultimately-trusted key against a backup copy on a write-protected floppy disk. It may be a good idea to do this hygienic maintenance periodically to make sure nothing is wrong with your public keyring.

To force PGP to perform a full analysis of your public keyring, use the **-kc** (keyring check) command:

pgp -kc

You can also use the following command to make PGP check all the signatures for a single selected public key:

pgp -kc <your_userID> [<keyring_filename>]

Revoking a key

By revoking a key, you are telling people that the key should no longer be used. You should revoke a public key if you think that its corresponding private key has been compromised.

To revoke a key, you need the private portion of the key and the passphrase. As a safeguard against the possibility of forgetting your passphrase or losing your private key, you might want to specify a designated revoker. For more information on specifying a designated revoker, see [“Adding a designated revoker to your key” on page 62](#).

NOTE: You cannot ‘unrevoke’ a key.

To revoke a key:

1. Ensure that the public portion of the key being revoked is on your keyring.
2. Enter the following on the command line:

```
pgp -kd <userID_of_key_being_revoked>
```
3. PGP asks if you want to permanently revoke the key. Type **y** to revoke the key.
4. If you are a designated revoker for this key, then you are prompted to enter your passphrase.

The best way to circulate a revoked key is to place it on a public key server.

Disabling a key

Sometimes you may want to temporarily disable a key. The ability to disable keys is useful when you want to retain a public key for future use, but you don't want it in your way when you perform encryption operations.

To disable a key:

1. Enter the following on the command line:

```
pgp -kd <userID_of_key_being_disabled>
```

If the key is implicitly trusted, PGP asks if you want to permanently revoke the key.

2. To disable the key (not permanently revoke the key), type **n**.
PGP asks if you want to disable the key.
3. Type **y** to disable the key.

Splitting and rejoining a key

Any private key can be split into shares among multiple “shareholders” using a cryptographic process known as Blakely-Shamir key splitting. This technique is recommended for extremely high security keys. For example, PGP Security, Inc. keeps a corporate key split between multiple individuals. Whenever we need to sign with that key, the shares of the key are rejoined temporarily.

Creating a split key

To create a split key, you are asked to specify the minimum number of people required to rejoin the key and the number of shares to make.

The resulting shares are saved as files either encrypted to the public key of a shareholder or encrypted conventionally if the shareholder has no public key. After the key has been split, the share files must be sent to the shareholders via ftp or email.

Attempts to sign or decrypt with a split key will automatically cause PGP to temporarily rejoin the key.

To create a split key:

1. Enter the following on the command line:

```
pgp -kl <userID_of_key_to_split>
```

2. Enter the user ID of the first shareholder.
3. Enter the number of shares for this user, or accept the default (one share).

Repeat steps 2 and 3 until you have specified all the shareholders, then hit **Return**.

4. Enter the minimum number of shares needed to rejoin the key, known as the threshold, or accept the default value.
5. Enter the passphrase of the key you are splitting.

PGP splits the key into the number of shares specified, then encrypts each portion of the key to the specified shareholder. Each shareholder receives a share file (in **.shf** format) for every share he/she owns.

For example, if you were to split a company signing key between 3 shareholders (**holder1**, **holder2**, and **holder3**) where **holder1** gets 2 shares, and the other recipients each get 1 share, the resulting share files would be:

```
<userID> <number_of_shares> <share_filename>
```

```
holder1 2 Shares.shf
```

```
holder2 1 Shares.shf
```

```
holder3 1 Shares.shf
```

To verify the key has been split, use the **-kv** (key view) option. The key displays “Split Key” before the user ID.

Reconstituting a split key

Once a key is split among multiple shareholders, attempting to sign or decrypt with it will cause PGP to automatically attempt to rejoin the key. There are two ways to rejoin the key, *locally* and *remotely*.

To rejoin key shares locally requires the shareholders presence at the rejoining computer. Each shareholder is required to enter the passphrase for his/her key share.

To rejoin key shares remotely requires the remote shareholders to authenticate and decrypt their keys before sending them over the network. PGP's Transport Layer Security (TLS) provides a secure link to transmit key shares which allows multiple individuals in distant locations to securely sign or decrypt with his/her key share.

Before receiving key shares over the network, you should verify each shareholder's fingerprint and sign his/her public key to ensure that the authenticating key is legitimate. To learn how to verify a key pair, see ["Validity and trust" on page 53](#).

Reconstituting a split key locally

To reconstitute a split key locally requires the shareholder's presence at the rejoining computer. Each shareholder must enter his/her own passphrase to decrypt the share file encrypted to his/her key.

To join a key locally:

1. Enter the following on the command line:

```
pgp -kj <userID_of_key_to_join>
```

2. Enter the share file of the first shareholder.

For example, you might enter **holder1 2 Shares.shf** from the previous example in ["Creating a split key" on page 65](#).

3. Enter the passphrase belonging to this shareholder.

The portion of the split key encrypted to this shareholder decrypts.

PGP displays the number of valid shares from each shareholder, as well as, the minimum number of shares needed to rejoin the key.

4. Repeat steps 2 and 3 until the minimum number of shares needed to rejoin the key is decrypted.

The key is rejoined.

5. Enter a new passphrase for the key.

Reconstituting a split key over the network

To reconstitute a split key over the network, you use the **-kq** option. Once you have created a split key, you must send the shares to the shareholders in other locations. You can do this by ftp or via email.

NOTE: You must have a signing key on your keyring to set up a TLS connection, which provides a secure link to transmit the key shares securely to individuals in other locations.

To join a key over the network:

From the system that contains the split key:

1. Enter the following on the command line:

```
pgp -kj <userID_of_key_to_join>
```
2. Press Enter.
3. PGP chooses a signing key on your keyring to set up a TLS connection.
4. Enter the passphrase for this key.
5. The system opens a TLS connection and waits to receive the shares.
The system displays, “Listening...”
6. When prompted, type **y** to confirm the connection.

From each remote site:

7. Enter the following on the command line:

```
pgp -kq <IP_address_of_system_with_split_key>
```
8. Enter the share file to send to the system with the split key.

For example, you might enter **holder1 2 Shares.shf** from the previous example in [“Creating a split key” on page 65](#).

9. Enter your passphrase to decrypt the share.
The system displays, “Preparing to send the key share.”
10. PGP chooses a signing key on your keyring to authenticate the TLS connection.
11. Enter the passphrase for this key.
12. When prompted, type **y** to confirm the connection.

From the system that contains the split key:

13. When prompted, type **y** to confirm the connection.
Once the connection is confirmed on both ends, the share is sent securely over the network and received by the system containing the split key. When the minimum number of shares needed to rejoin the key is received, then the key is rejoined.
14. Enter a new passphrase for the key.

Additional Decryption Keys

Suppose your chief scientist is hit by a bus and is hospitalized for months. Or that your lead engineer, in a rage, encrypts his entire hard drive and leaves the company. What happens to all that data, which is so securely encrypted? Can you retrieve it, or is it gone forever?

An *Additional Decryption Key* (ADK) is a data recovery tool. In an environment that enforces use of an ADK, any information encrypted to a user’s key is also encrypted to the Additional Decryption Key. When someone inside or outside the organization encrypts information to a user, the information is also encrypted to the Additional Decryption Key. This allows the owner of the Additional Decryption Key to decrypt any information sent to the user. This process happens automatically and is fully integrated into the encryption process.

Recover data in an emergency

An ADK is a powerful security tool in situations where an employee is injured, incapacitated, or terminated, leaving valuable information encrypted. Because PGP has no “back door,” recovery of this information would be otherwise infeasible.

While you may not ordinarily use your ADKs, there may be circumstances when it is necessary to recover someone’s data, for example, if someone is out of work for some time or if you are subpoenaed by a law enforcement agency and must decrypt messages or files for a court case.

Data recovery versus key recovery

Do not confuse data recovery with key recovery. An Additional Decryption Key lets you recover information that has been encrypted to a particular key, not the key itself. The difference is crucial. If a mechanism exists to obtain a copy of a user’s key, one major feature of a public-key cryptosystem—non-repudiation—is lost. If more than one copy of a key exists, then a user can deny having signed information with the key.

Retaining copies of users’ keys has an added security risk: the machine storing the keys is an obvious target for attack, as is the administrator of the machine.

An Additional Decryption Key is far easier to protect, and it enables you to retain non-repudiation, which is a major advantage inherent to public-key cryptography.

Types of ADKs

PGP offers two types of ADKs: Incoming ADKS, and Outgoing ADKs.

- An *incoming* ADK is used by PGP during key generation. An incoming ADK’s key ID is associated with new keys during key generation, and henceforth when someone attempts to encrypt to the new key, PGP also attempts to encrypt to the ADK. Incoming ADKs may be either Diffie-Hellman/DSS or RSA keys. You cannot use an RSA Legacy key as an incoming ADK.
- An *outgoing* ADK is associated with an installation of PGP. Outgoing ADKs are automatically added to users’ keyrings and are always part of a recipient list. Outgoing ADKs can be of any key type.

Additional Decryption Key policy

Your Security Officer must decide whether your company enforces the use of ADKs. You should have a policy that governs how and when they will be used and should communicate this policy to everyone who will be affected by it. Obviously, this policy should consider employee privacy.

Protecting your Additional Decryption Key

Additional Decryption Keys must be secured both physically and electronically in order to prevent a security breach. If either the incoming or outgoing ADK is ever compromised, all encrypted messages sent to users with additional decryption enabled could be decrypted by the attacker.

To prevent unauthorized additional decryption and problems with liability, your organization should enforce a policy that the key should be split and shared by at least two individuals.

IMPORTANT: Do *not* use ADKs unless you can ensure their security. In an environment that enforces use of an ADK, security of these keys determines the security of all encrypted messages in your entire organization.

Implementing your Additional Decryption Keys

To implement ADKs in your environment, you must first create the ADK(s).

NOTE: If you want separate keys for the incoming ADK and the outgoing ADK, you must go through the Key Generation process twice, once for each key.

1. Do one of the following:
 - Set the incoming ADK by specifying the **ADKKEY** parameter in the configuration file, and then generate a key (the **-kg** option) that meets your needs in terms of key type and key size.

or

- Enter the following on the command line:

```
pgp +ADKKEY=<keyID> -kg
```

2. If you want to enforce use of the ADK, set the **ENFORCEADK** parameter to **ON**.

For more information on setting the **ADKKEY** configuration parameter, see “[ADKKEY](#)” on page 84. For more information on setting the **ENFORCEADK** configuration parameter, see “[ENFORCEADK](#)” on page 90.

Deleting a key from a key server

If the key server permits key removal, you can delete your own key. Bear in mind that the key may reappear on the server; if other users upload your key to the server for example.

```
pgp -kr <userID_of_key_to_remove> <keyserver_URL>
```

An example of a **URL** is **ldap://certserver.pgp.com**

This chapter describes several advanced PGP topics and commands.

To:	See:
Use scripts with PGP	“Using scripts with PGP” on page 73
Encrypt and transmit binary data	“Encrypting and transmitting binary data” on page 76
Send ASCII files to different machine environments	“Working with ASCII and binary data” on page 76
Wipe your disk	“Wiping your disk” on page 78
Work with passphrases	“Alternative ways to work with passphrases” on page 79
Work with groups	“Working with groups” on page 80

Using scripts with PGP

To use PGP without interaction requires you to create a script using PGP commands.

You can run PGP in “batch” mode (for example, from an MSDOS “.bat” file or from a UNIX shell script). **MSDOS** refers to the Windows NT command prompt.

For example, if you wanted to create a script to encrypt and sign several files inside a loop, then you might include the following syntax:

```
pgp +batchmode +force -es $filename $userID -z $passphrase
```

PGP Command Line supports limited batch mode capabilities. You can eliminate certain questions using the **FORCE** variable, such as “Do you want to override a file with the same name?” You can also cause PGP to automatically respond with its own default answers using the **BATCHMODE** variable. Note, however, that you cannot alter PGP’s responses.

Suppressing unnecessary questions

Use the **BATCHMODE** option to eliminate unnecessary questions.

BATCHMODE

When the **BATCHMODE** flag is enabled on the command line, PGP does not ask any unnecessary questions or prompt for alternate filenames. However, note that with **BATCHMODE**, PGP will respond with *its own* default responses. You cannot alter the responses.

Syntax

```
pgp +batchmode < ciphertext_filename >
```

This variable is useful when you run PGP from shell scripts or batch files. When **BATCHMODE** is on, some key management commands still need user interaction (for example, if a passphrase is required on a key), so shell scripts may need to avoid them.

You can also enable **BATCHMODE** to check the validity of a signature on a file:

- If there was no signature on the file, the exit code is **1**.
- If there was a good signature on the file, the exit code is **0**.

Eliminating confirmation questions

Use the **FORCE** option to eliminate confirmation questions.

FORCE

To run PGP non-interactively from a UNIX shell script or MSDOS batch file, you can use the **FORCE** option to eliminate interaction with PGP in the following two situations.

- When you decrypt a file that has a filename with the same name as another in the directory, **FORCE** causes PGP to overwrite the original file without prompting.
- When you remove a key from a keyring (either public or private), **FORCE** removes the key without confirming the deletion.

To use the **FORCE** option, use the following syntax on the command line:

```
pgp +force <ciphertext_filename>
```

or:

```
pgp -kr +force <your_userID>
```

Default Value

FORCE = off

Understanding PGP exit status codes

When you run PGP in batch mode, PGP returns an error exit status to the shell.

- A zero exit status code signifies a normal exit.
- A non-zero exit status code tells you that an error occurred. Different error exit conditions return different exit status codes to the shell.

For a list of PGP exit codes, see [Appendix A, “Exit and Error Codes”](#).

Using PGP as a UNIX-style filter

UNIX uses pipes to make two applications work together. The output of one application can be directly fed through a pipe to be read as input to another application. For this to work, the applications must be capable of reading the raw material from “standard input” and writing the finished output to “standard output.”

To use PGP’s UNIX-style filter mode, reading from standard input and writing to standard output, add the **-f** option:

```
pgp -feast <recipients_userid> < <input_filename> > <output_filename>
```

For example:

```
pgp -feast tsmith < confidential.doc > confidential.pgp
```

This feature makes it easier to use PGP with scripts and email applications.

When you use PGP’s filter mode to decrypt a ciphertext file, you may find the **PGPPASS** environment variable useful. This variable holds the passphrase so that PGP does not prompt you for this information. For more information, see [“Storing your passphrase with PGPPASS” on page 79](#).

Working with ASCII and binary data

ASCII-armored text is binary data that has been encoded using a standard, printable, 7-bit ASCII character set. This allows users to transport the information through many email systems that only allow messages that contain ASCII text.

The sections to follow describe how to:

- encrypt and transmit binary data
- send binary data files in ASCII-armored format without encryption or signature
- decrypt ASCII-armored messages
- send a public key in ASCII-armored format
- send ASCII text files to different machine environments

Encrypting and transmitting binary data

Many email systems only allow messages that contain ASCII text. As a result, PGP supports an ASCII-armored format for ciphertext messages (similar to MIME).

This format, which represents binary data using only printable ASCII characters, enables you to transmit binary encrypted data through 7-bit channels, or to send binary encrypted data as normal email text. PGP's ASCII-armored format acts as a form of "transport armor," protecting the message against corruption as it travels through intersystem gateways on the Internet. PGP also appends a CRC to detect transmission errors.

ASCII-armored format converts the plaintext by expanding groups of 3 binary 8-bit bytes into 4 printable ASCII characters. As a result, the file expands by about 33%. However, this expansion is offset by the compression that occurs before encryption.

To produce an ASCII-armored formatted file, enter the following command:

```
pgp -ea <plaintext_filename> <recipient's_userID>
```

This command instructs PGP to produce a ciphertext file in ASCII-armored format called **.asc**. This file contains data in a MIME-like ASCII-armored format. You can upload the file into a text editor through 7-bit channels and transmit as normal email.

Sending binary data files in ASCII-armored format without encryption or signature

Use PGP's **-a** option to convert a file into ASCII-armored format. No encryption or signing is involved, so neither sender nor recipient requires a key. When you use the **-a** option, PGP attempts to compress the data before converting it to ASCII-armored format. Use the command as follows:

```
pgp -a <binary_filename>
```

This command instructs PGP to produce an ASCII-armored file called **filename.asc**. The recipient uses the **-p** option to unwrap the message and restore the sender's original filename.

Decrypting ASCII-armored messages

To decrypt an ASCII-armored message, enter the following command:

```
pgp <ASCII-armored_filename>
```

PGP recognizes that the file is in ASCII-armored format, converts the file back to binary, and creates an output file in normal plaintext form.

When PGP is decrypting the message, it ignores any extraneous text in mail headers that are not enclosed in the ASCII-armored message blocks.

Sending a public key in ASCII-armored format

To send a public key to someone else in ASCII-armored format, add the **-a** option while extracting the key from your keyring.

```
pgp -kxa <userID> <key_filename> [<keyring>]
```

If you forgot to use the **-a** option when you made a ciphertext file or extracted a key, you can convert the binary file into ASCII-armored format by using the **-a** option (do not specify encryption). PGP converts the file to a **".asc"** file.

Sending ASCII text files to different machine environments

PGP encrypts any plaintext file, binary 8-bit data, or ASCII text. The most common use of PGP is for email, which is ASCII text.

ASCII text is represented differently on different machines. For example, on an MSDOS system, all lines of ASCII text are terminated with a carriage return followed by a linefeed. On a UNIX system, all lines end with just a linefeed. On a Macintosh, all lines end with just a carriage return.

Normal unencrypted ASCII text messages are often automatically translated to some common “canonical” form when they are transmitted from one machine to another. Canonical text has a carriage return and a linefeed at the end of each line of text.

Encrypted text cannot be automatically converted by a communication protocol because the plaintext is hidden by encipherment. To remedy this problem, PGP’s **-t** option lets you specify that the plaintext be treated as ASCII text and converted to canonical text before encryption. When the message is received, the decrypted plaintext is automatically converted to the appropriate text form for the local environment.

To use this feature, enter the **-t** option when encrypting or signing a message:

```
pgp -et <plaintext_filename> <recipient's_userID>
```

PGP includes an environment variable that corresponds to the **-t** option, **TEXTMODE**. If you consistently receive plaintext files rather than binary data, set **TEXTMODE=on** in the PGP configuration file. For more information on setting configuration parameters, see [Chapter 8, “PGP’s Configuration File”](#).

Wiping your disk

After PGP produces a ciphertext file for you, you can request PGP to automatically overwrite and delete the plaintext file, leaving no trace of plaintext on the disk. Use the **-w** (wipe) option when a plaintext file contains sensitive information; it prevents someone from recovering the file with a disk block scanning utility.

Use the **-w** (wipe) option when you encrypt and sign a message:

```
pgp -ew <plaintext_filename> <recipient's_userID>
```

For example:

```
pgp -ew confidential.txt mjohnson
```

This instructs PGP to create a ciphertext file **confidential.pgp** and to destroy the plaintext file **confidential.txt**.

Note that this option will not wipe out any fragments of plaintext that your word processor might have created on the disk while you were editing the message before running PGP. Most word processors create backup files, scratch files, or both.

By default, PGP overwrites the file three times.

Alternative ways to work with passphrases

PGP generally prompts you for your passphrase. If you want to streamline your interaction with PGP, you can use one of the following methods for supplying PGP with your passphrase.

NOTE: The environment variable method of defining a file descriptor for supplying PGP with your passphrase is not the most secure method.

The recommended method is to set the **PASSPHRASE-FD** variable in the PGP configuration file. For more information on setting the **PASSPHRASE-FD** variable, see [“Specifying configuration values” on page 83](#).

Storing your passphrase with PGPPASS

WARNING: You should not use this feature if working on a shared system. The passphrase may be visible to others.

When PGP needs a passphrase to unlock a secret key, PGP prompts you to enter your passphrase. Use the **PGPPASS** environment variable to store your passphrase. When PGP requires a passphrase, it attempts to use the stored passphrase. If the stored passphrase is incorrect, PGP recovers by prompting you for the correct passphrase.

SET PGPPASS= *<passphrase>*

The following is an example of how you might set this variable in the environment.

SET PGPPASS=“zaphod beeblebrox for president”

The above example would eliminate the prompt for the passphrase if the passphrase was “*zaphod beeblebrox for president*”.

This feature is convenient if you regularly receive a large number of incoming messages addressed to your secret key, eliminating the need for you to repeatedly type in your passphrase.

The recommended way to use this feature is to enter the command each time you boot your system, and erase it or turn off your machine when you are done.

Passing your passphrase from another application

PGP includes a command line option, **-z**, that you can use to pass your passphrase into PGP from another application. This option is designed primarily to invoke PGP from inside an email package.

The passphrase follows the **-z** option on the command line. Use this feature with caution.

For example:

```
pgp -s <filename> -z <passphrase>
```

PGPPASSFD

Use the **PGPPASSFD** (PGP passphrase file descriptor) environment variable to supply PGP with the file descriptor to which the passphrase will be passed. This is most useful when writing scripts. This parameter cannot be used if more than one passphrase must be supplied.

```
SET PGPPASSFD=<file_descriptor_number>
```

If this environment variable is set to zero (0), the passphrase is read from standard input (STDIN). PGP uses the first text line from the specified filename as the password.

NOTE: A **PASSPHRASE-FD** value specified in the configuration file supersedes a value set in the **PGPPASSFD** environment variable. For more information on setting the **PASSPHRASE-FD** value, see [“Specifying configuration values” on page 83](#).

Working with groups

You may find that you need to perform encryption operations to multiple people at one time. Specifying them individually is inefficient. Instead, you can create distribution lists, or groups, that include everyone to whom you want to encrypt.

For example, if you want to encrypt a file to 10 people at HRdepartment@nai.com, you would create a distribution list of that name. You would then add the keys for all 10 members of the HR department mailing list to the group. This enables you to encrypt a file to all 10 people in a single operation.

The **-g** (group help) option displays help on all group options.

Creating a group

To create a group, you use the **-ga** (group add) option. This option adds a group definition to the groups file (**pgpgroup.pgr**).

```
pgp -ga <groupname>
```

The following syntax creates a group with the name “engineers.”

```
pgp -ga engineers
```

Add recipients to a group (the -ga option)

You can also add users (or groups) to groups using the **-ga** option.

```
pgp -ga <groupname> <userID>
```

The following syntax adds the key for Cal Pettit to the **engineers** group.

```
pgp -ga engineers “Cal Pettit”
```

Viewing a group (the -gv and -gvv options)

The **-gv** (group view) and **-gvv** (group view and content) options enable you to obtain more information about your groups.

```
pgp -gv <groupname> shows the name and description of the group.
```

```
pgp -gvv <groupname> shows the contents of the specified group. (If you do not specify a group, -gvv shows the contents of all of your groups.)
```

Remove recipients from a group (the -gr option)

To remove members or groups from a group, use the **-gr** (group remove) option.

```
pgp -gr <groupname> <userID>
```

The following syntax removes the key for Cal Pettit from the **engineers** group.

```
pgp -gr engineers “Cal Pettit”
```


Learning about PGP's configuration file

PGP stores a number of user-defined parameters in the configuration text file **pgp.cfg**. A configuration file enables you to define flags and parameters for PGP, eliminating the need to define these parameters at the command line. (For more information on **pgp.cfg** and its location, see [“Setting the location of PGP files” on page 23.](#))

Use these configuration parameters to perform the following tasks as well as many others:

- Control where PGP stores its temporary scratch files.
- Adjust PGP's level of skepticism when it evaluates a key's validity based on the number of the key's certifying signatures.
- Set the location and name of your keyrings.

Specifying configuration values

Configuration parameters may be assigned integer values, character string values, or on/off values; the type of values depends on the type of parameter. PGP includes a sample configuration file for your review.

The following rules apply to the configuration file:

- PGP ignores blank lines.
- PGP also ignores characters that follow the comment character, #.
- Keywords are not case-sensitive.

The following is a short sample fragment of a typical configuration file, where the file's owner used comments in conjunction with the actual settings:

```
# TMP is the directory for PGP scratch files, such as a RAM disk.
TMP = "e:\\"      # Can be overridden by environment variable TMP.
Armor = on       # Use -a flag for ASCII armor whenever applicable.
# CERT_DEPTH is how deeply introducers may introduce introducers.
cert_depth = 3
```

PGP uses default values for the configuration parameters under the following conditions:

- When configuration parameters are not defined.
- If the configuration file does not exist.
- If PGP cannot find the configuration file.

Setting configuration parameters from the command line

You can also set configuration parameters directly from the PGP command line. You must precede the parameter setting with a “+” (plus) character.

For example, the following two PGP commands produce the same effect:

```
pgp -e +armor=on message.txt smith
```

```
pgp -ea message.txt smith
```

For the location of the `pgp.cfg` file, please refer to [“Setting the location of PGP files” on page 23](#).

The remainder of this chapter summarizes PGP's configuration parameters in alphabetical order.

Configuration parameters

ADKKEY

Encrypt to an Additional Decryption Key (ADK). When this parameter is used, all generated keys have an ADK equal to the value of **ADKKEY**, and everything encrypted to the user's key is also encrypted to the ADK key identified by this parameter. Note the difference between *incoming* ADKs and *outgoing* ADKs as described in [Chapter 6, “Key Management Operations”](#).

If you choose to use two different keys for the incoming and outgoing ADKs, you can set **ADKKEY** to specify one of the keys, then use the **+ADDKEY** command to override it during key generation.

Default Value

```
ADKKEY=""
```

Syntax

```
ADKKEY = <keyID>
```

For example, **ADKKEY = “0xAB12C34D”**

Notes

You use **ADKKEY** in conjunction with the parameter **ENFORCEADK** to determine whether PGP enforces the use of ADKs. If **ENFORCEADK** is not set, then users can subvert use of the ADK.

If **ENFORCEADK** is **on** and the encryption key was generated with **ENFORCEADK** set to **on**, data is always encrypted to the ADK if the ADK key is available. If the ADK key is not available, an error message displays and the encryption operation fails.

If **ENFORCEADK** is set to **off** and the ADK key is not present on the user's keyring, PGP displays a warning message and does not encrypt to the ADK key.

You can also set this parameter on the command line with the **+** prefix; for example, **+ADKKEY="0xAB12C34D"**.

NOTE: We recommend you always specify the **ADKKEY** in your configuration file using the key ID to prevent any potential security holes—if you were to specify the **ADKKEY** using the key's user ID, an interloper might create another key with the same user ID and introduce a means for decrypting secret data.

ARMOR

If enabled, this parameter causes PGP to emit ciphertext or keys in ASCII-armored format suitable for transport through email channels.

Default Value

ARMOR = off

Notes

- Output files are named with the **.asc** extension.
- The configuration parameter **ARMOR** is equivalent to the **-a** command line option.
- If you intend to use PGP primarily for email purposes, you should turn this parameter on (**ARMOR=on**).

BATCHMODE

When the **BATCHMODE** flag is enabled on the command line, PGP does not ask any unnecessary questions or prompt for alternate filenames. With **BATCHMODE**, PGP will respond with *its own* default responses. You cannot alter the responses.

Default Value

BATCHMODE = off

Syntax

When you use this option from the command line, the syntax is as follows:

```
pgp +batchmode < ciphertext_filename >
```

This variable is useful when you run PGP from shell scripts or batch files. When **BATCHMODE** is on, some key management commands still need user interaction (for example, if a passphrase is required on a key), so shell scripts may need to avoid them.

You can also enable **BATCHMODE** to check the validity of a signature on a file:

- If there was no signature on the file, the exit code is **1**.
- If there was a good signature on the file, the exit code is **0**.

CERT_DEPTH

The configuration parameter **CERT_DEPTH** identifies how many levels deep you can nest trusted introducers. (Trusted introducers are those people who you trust to certify—or validate—others' keys. If a trusted introducer certifies a key, it will appear valid on your public keyring.)

Default Value

CERT_DEPTH = 4

Notes

For example, if **CERT_DEPTH** is set to **1**, there can only be one layer of introducers below your own ultimately-trusted key. If that is the case, you are required to directly certify the public keys of all trusted introducers on your keyring. If you set **CERT_DEPTH** to zero, you could have no introducers at all, and you would have to directly certify each and every key on your public keyring to use it.

The minimum **CERT_DEPTH** is **0**; the maximum is **8**.

CIPHERNUM

Specifies which symmetric cipher PGP should use to encrypt the session key—IDEA, Triple-DES, CAST, AES, or Twofish.

Default Value

CIPHERNUM = 1

Values are as follows:

IDEA = 1

3DES = 2

CAST5 = 3

AES128 = 7

AES192 = 8

AES256 = 9

Twofish = 10

CLEARSIG

Use the **CLEARSIG** parameter to generate a signed message that can be read with human eyes, without the aid of PGP. The recipient must still use PGP to verify the signature. For an example of a clear-signed message, see [“Producing a clear-signed message” on page 49](#).

Unencrypted PGP signed messages have a signature certificate prepended in binary form. The signed message is compressed, rendering the message unreadable to human eyes, even though the message is not encrypted.

To send this binary data through a 7-bit email channel, PGP applies ASCII-armor (see the **ARMOR** parameter). Even if PGP did not compress the message, the ASCII armor renders the message unreadable to human eyes. The recipient must first use PGP to strip the armor off the message, and then decompress the message before reading it.

If the original plaintext message is in text, not binary form, you can use the **CLEARSIG** parameter to send a signed message through an email channel; the signed message is not compressed, and the ASCII armor is applied to the binary signature certificate, but not to the plaintext message. The **CLEARSIG** parameter makes it possible to generate a signed message that can be read with human eyes, without the aid of PGP (again, the recipient still needs PGP to verify the signature).

Default Value

CLEARSIG = on

Notes

- To enable the full **CLEARSIG** behavior, the **ARMOR** and **TEXTMODE** flags must also be turned on. Set **ARMOR=ON** (or use the **-a** option), and set **TEXTMODE=ON** (or use the **-t** option). If **CLEARSIG** is set to off in your configuration file, you can turn it back on again directly on the command line:

pgp -sta +clearsig=on message.txt

- Note that since this method only applies ASCII armor to the binary signature certificate, and not to the message text itself, there is some risk that the unarmored message may suffer some accidental molestation while en route. This can happen if it passes through an email gateway that performs character set conversions, or in some cases extra spaces may be added to or stripped from the ends of lines. If this occurs, the signature will fail to verify, which may give a false indication of intentional tampering.
- When PGP calculates the signature for text in **CLEARSIG** mode, trailing blanks are ignored on each line.

COMMENT

Displays a comment header in all armored output just beneath the PGP Version header.

Default Value

COMMENT = ""

COMPATIBLE

The configuration parameter **COMPATIBLE** enables user-interface compatibility with PGP 2.6.2. You may require this feature for interoperation with scripts that parse the output or otherwise interact with PGP dialogs.

Default Value

COMPATIBLE=off

COMPLETES_NEEDED

The configuration parameter **COMPLETES_NEEDED** identifies the minimum number of completely trusted introducers required to fully certify a public key on your public keyring. For more information on trusted introducers, see *An Introduction to Cryptography*.

Default Value

COMPLETES_NEEDED = 1

COMPRESS

The configuration parameter **COMPRESS** enables or disables data compression before encryption. It is used mainly to debug PGP. Under normal circumstances, PGP attempts to compress the plaintext before it encrypts it. Compression strengthens security. Turning **COMPRESS** off therefore weakens your security. Thus, we recommend you do not change this setting.

Default Value

COMPRESS = on

ENCRYPTTOSELF

Instructs PGP to always add the recipient specified in the configuration parameter **MYNAME** to its list of recipients and thus always encrypt to the predefined key as well as to any specified recipients.

NOTE: Just because you originated the encryption does not mean you can decrypt the information. If you want to have access later to messages you encrypt to another person, you must enable **ENCRYPTTOSELF**.

Default Value

ENCRYPTTOSELF = off

ENFORCEADK

Forces encryption to any ADKs associated with a recipient's key.

Default Value

ENFORCEADK = off

Use in conjunction with the **ADKKEY** parameter (encrypting to a key with an Additional Decryption Key (ADK)). For more information on ADKs, see [“Implementing your Additional Decryption Keys” on page 71](#).

If a user tries to encrypt to a key that is associated with an ADK and **ENFORCEADK** is set to **on**, PGP attempts to encrypt to the ADK as well. If the ADK is not present on the keyring, PGP generates an error message.

If **ENFORCEADK** is set to **off** and the ADK is not present on the user's keyring, PGP displays a warning message and does not encrypt to the ADK key.

You can set this parameter on the command line using the **+** prefix, for example, **+ENFORCEADK=on**.

EXPORTABLE

This is the equivalent of setting **SIGTYPE=export**. This makes the most sense if the **SIGTYPE** parameter is set to **non** in the configuration file, and then you use **+exportable** on the command line to override the non-exportable signature type. For more information on the **SIGTYPE** parameter and settings, see [“SIGTYPE” on page 98](#).

Default Value

EXPORTABLE=on

FASTKEYGEN

Use to specify fast key generation.

Default Value

FASTKEYGEN = on

FORCE

To run PGP non-interactively from a UNIX shell script or MSDOS batch file, you can use the **FORCE** option to eliminate interaction with PGP in the following two situations.

- When you decrypt a file that has a filename with the same name as another in the directory, **FORCE** causes PGP to overwrite the original file without prompting.
- When you remove a key from a keyring (either public or private), **FORCE** removes the key without confirming the deletion.

Default Value

FORCE=off

Notes

When you use this option from the command line, the syntax is as follows:

```
pgp +force <ciiphertext_filename>
```

or:

```
pgp -kr +force <your_userID>
```

GROUPSFILE

Specifies the location of the PGP groups file, **pgpgroup.pgr**.

Default Value

Unix

```
GROUPSFILE = "<HOME>/pgp/pgpgroup.pgr"
```

Windows NT/2000

```
GROUPSFILE = "<USERPROFILE>\Application Data\pgp\pgpgroup.pgr"
```

Notes

The **<HOME>** and **<USERPROFILE>** portions of the paths must be replaced with the value of the current environment variables of the same names.

HASHNUM

Defines which hash algorithm PGP uses for signing.

Default Value

HASHNUM= 1

Values are as follows:

MD5 = 1

SHA-1 = 2

RIPEND160 = 3

INTERACTIVE

Instructs PGP to ask for confirmation when you add a key file with multiple keys to your keyring. When this variable is set to “**on**”, PGP asks for confirmation for each key in the key file before adding it to your keyring.

Default Value

INTERACTIVE = off

MARGINALS_NEEDED

The configuration parameter **MARGINALS_NEEDED** identifies the minimum number of marginally trusted introducers required to fully certify a public key on your public keyring. For more information on trusted introducers, see *An Introduction to Cryptography*.

Default Value

MARGINALS_NEEDED = 2

MYNAME

The configuration parameter **MYNAME** specifies the default key ID to use when selecting a private key for making signatures. If **MYNAME** is not defined, PGP uses the most recent private key you installed on your private keyring (**secring.skr**). You can override this setting by using the **-u** option to specify a user ID on the PGP command line.

Default Value

MYNAME = ""

Notes

- **ENCRYPTTOSELF** refers to **MYNAME**.
- You should always specify **MYNAME** using the key's key ID, not user ID, to prevent a potential security risk.

PAGER

PGP's **-m** option lets you view decrypted plaintext output on your screen, one screen at a time, without writing the output to a file. If you want to be able to page backwards, one screen at a time, set **PAGER=less**, and use the PGP **-m** option.

Default Value

PAGER = ""

PGP includes a built-in page display utility. If you prefer to use a different page display utility, use the **PAGER** parameter to identify the utility. The **PAGER** parameter specifies the shell command PGP uses to display a file.

Note that if the sender specified that a file is "for your eyes only," PGP always uses its own built-in display function.

For further details, see ["Viewing decrypted plaintext output on your screen" on page 47](#).

PASSPHRASE-FD

If **PASSPHRASE-FD** is specified, then PGP will try to read the passphrase from the stipulated file descriptor. Use this parameter to transmit your passphrase from one program to another. Set the **PASSPHRASE-FD** parameter equal to a file descriptor number.

Default Value

PASSPHRASE-FD=

Notes

If **PASSPHRASE-FD** is not specified in the configuration file (and the **PGPPASSFD** environment variable is also not set), then the passphrase is read from standard input (STDIN).

The **PASSPHRASE-FD** value specified in the configuration file supersedes a value set in the **PGPPASSFD** environment variable. For more information on your passphrase options, see [“Alternative ways to work with passphrases” on page 79](#).

PASSTHROUGH

When set to **on**, PGP does not generate errors when a decryption operation encounters an unrecognized lexical section.

Default Value

PASSTHROUGH=off

PGP_MIME

Use to specify compatibility with PGP/MIME. The **PGP_MIME** parameter creates messages in PGP/MIME format.

PGP/MIME format produces both a message “header” and a message “body.” The **PGP_MIME** parameter causes the output to include both of these parts, separated by a blank line.

This parameter should be used in conjunction with the **ARMOR** parameter.

Default Value

PGP_MIME = off

PGP_MIMEPARSE

Use to instruct PGP to try to parse MIME body parts. To receive messages in PGP/MIME format properly, both the message “header” and message “body” need to be input to the program.

In some cases, the mail client does not save the message header. Use **PGP_MIMEPARSE** to tell PGP that only the message body is being provided as input, and the header is missing. PGP attempts to parse the message body and find the PGP/MIME information even though the header is missing.

NOTE: This method is not completely reliable at this time. It is preferable that you include the MIME headers whenever possible.

Default Value

PGP_MIMEPARSE = off

PUBRING

You may want to keep your public keyring in a directory separate from your PGP configuration file (that is, the directory specified by your **PGPPATH** environment variable). Use the **PUBRING** parameter to identify the full path and filename for your public keyring.

Default Value

Unix

PUBRING = “<PGPPATH>/pubring.pkr”

For example:

PUBRING = ~/mykeyrings/pubkeys.pkr

Windows NT/2000

PUBRING = “<USERPROFILE>\Application Data\pgp\pubring.pkr”

For example:

PUBRING = c:/personal/keyrings/public_keys.pkr

Notes

The **<PGPPATH>** and **<USERPROFILE>** portions of the paths must be replaced with the value of the current environment variables of the same names.

You can also use this feature on the command line to specify an alternative keyring. For example, you might set the value of **PUBRING** as shown below.

```
pgp +pubring=pubkeys.pkr -kv
```

RANDOMDEVICE

(UNIX only.) Identifies the system entropy pool, **/dev/random**. PGP tries to open this device to acquire entropy, and if that fails, will try to acquire entropy from user keystrokes.

Default Value

```
RANDOMDEVICE = /dev/random
```

RANDSEED

The random number seed file, **randseed.rnd**, is used to generate session keys. You may want to keep your random number seed file in a more secure directory or device (this file generally resides in the directory specified by your **PGPPATH** environmental variable). Use the **RANDSEED** parameter to identify the full path and filename for your random seed file.

Default Value

Unix

```
RANDSEED = "<PGPPATH>/randseed.rnd"
```

Windows NT/2000

```
RANDSEED = "<ALLUSERSPROFILE>\Application Data\ Network Associates\pgp\randseed.rnd"
```

Notes

The **<PGPPATH>** and **<ALLUSERSPROFILE>** portions of the paths must be replaced with the value of the current environment variables of the same names.

RSAVER

Specifies which version of RSA keys to use—RSA keys or RSA Legacy keys.

Default Value

RSAVER=3

- **RSAVER=3**
RSA Legacy keys. Enables RSA v3 key generation. RSA Legacy keys are compatible with previous versions of PGP. RSA Legacy keys do not allow encryption subkeys and cannot be used as incoming ADKs.
- **RSAVER=4**
Enables RSA v4 key generation. RSA v4 keys support subkeys and can be used as incoming ADKs. RSA v4 keys support all the features of Diffie-Hellman/DSS keys.

SECRING

You may want to keep your secret keyring in a directory separate from your PGP configuration file (that is, the directory specified by your **PGPPATH** environmental variable). Use the **SECRING** parameter to identify the full path and filename for your secret keyring.

Default Value

Unix

SECRING = "<PGPPATH>/secring.skr"

Windows NT/2000

SECRING = "<USERPROFILE>\Application Data\pgp\secring.skr"

Notes

The **<PGPPATH>** and **<USERPROFILE>** portions of the paths must be replaced with the value of the current environment variables of the same names. For example, you might set the value of **SECRING** as shown below.

Unix

SECRING = ~/mykeyrings/secring.skr

Windows NT/2000

SECRING = c:/personal/keyrings/secring.skr

SHOWPASS

Causes PGP to echo your typing during passphrase entry.

Default Value

SHOWPASS = off

PGP, by default, does not let you see your passphrase as you type it. This makes it harder for someone to look over your shoulder while you type and learn your passphrase. However, you may have problems typing your passphrase without seeing what you are typing or you may feel confident that you have sufficient privacy and do not need to hide your keystrokes.

SIGTYPE

Applies a type to a signature on a key. Signature types are discussed below.

Default Value

SIGTYPE=export

Syntax

pgp [+sigtype=export | non | meta | trusted] -ks <their_userID> [-u <your_userID>] [<keyring>]

- **Export.** Exportable signatures can be exported to a certificate server so other users can view them.
- **Non.** Non-exportable signatures apply only to your keyring. You cannot export non-exportable signatures to a certificate server.
- **Meta.** Meta signatures (always non-exportable) bestow meta-introducer status on the key. Any key considered trusted by the meta-introducer is considered a trusted introducer by you, and any key considered valid by the trusted introducer is considered valid to you.
- **Trusted.** Trusted signatures bestow trusted introducer status on the key. Any key considered valid by a trusted introducer is considered valid to you. When you apply a 'trusted' type to a signature on a key, PGP prompts you to enter a domain in which the key is trusted.

STATUS-FD

Use the **STATUS-FD** (status file descriptor) variable to write status messages to an alternative output stream. This parameter allows status messages to be redirected to a file descriptor. Set the **STATUS-FD** parameter equal to a file descriptor number.

Default Value

STATUS-FD =

TMP

Specifies the directory PGP uses for temporary scratch files. If **TMP** is undefined, the temporary files are written in the current directory. If the shell environmental variable **TMP** is defined, PGP stores temporary files in the named directory.

Default Value

TMP = ""

TEXTMODE

Causes PGP to assume the plaintext is a text file, not a binary file, and converts the plaintext to *canonical text* before encrypting it. Canonical text has a carriage return and a line feed at the end of each line of text.

Default Value

TEXTMODE = off

Notes

The configuration parameter **TEXTMODE** is equivalent to the **-t** command line option.

If you intend to use PGP primarily for email purposes, you should set **TEXTMODE=on**.

For further details, see [“Sending ASCII text files to different machine environments” on page 77](#).

TZFIX

The configuration parameter **TZFIX** specifies the number of hours to add to the system time function to get **GMT**. If your operating system does not give time in **GMT**, use **TZFIX** to adjust the system time to **GMT**.

Default Value

TZFIX = 0

Notes

PGP includes timestamps for keys and signature certificates in Greenwich Mean Time (GMT). When PGP asks the system for the time of day, the system should give the time in GMT. However, on some improperly configured systems, the system time is returned in US Pacific Standard Time time plus 8 hours.

You should make sure your system's time zone is configured correctly by setting the TZ environment variable.

For Los Angeles: SET TZ=PST8PDT

For Denver: SET TZ=MST7MDT

For Arizona: SET TZ=MST7

(Arizona does not use daylight savings time)

For Chicago: SET TZ=CST6CDT

For New York: SET TZ=EST5EDT

For London: SET TZ=GMT0BST

For Amsterdam: SET TZ=MET-1DST

For Moscow: SET TZ=MSK-3MSD

For Auckland: SET TZ=NZT-13

VERBOSE

The **VERBOSE** variable controls the amount of detail you receive from PGP diagnostic messages.

Default Value

VERBOSE = 1

Notes

The settings are as follows:

0 - Displays only queries and errors (that is, prompts the user for input and displays errors when they occur).

1 - Normal default setting. Displays a reasonable amount of detail in diagnostic or advisory messages.

2 - Displays maximum information, usually to help diagnose problems in PGP. Not recommended for normal use.

WITH-COLONS

The **WITH-COLONS** variable controls how the output of a key view (-kv) is formatted.

Default Value

WITH-COLONS = off

Notes

When you use the **WITH-COLONS** variable when viewing public and public/private key pairs, the output displays in the following format.

**[pub|sec]:validity:key_size:key_algorithm:keyid:creation_date:
expiration_date:trust:username:**

Abbreviation descriptions:

pub = public key

sec = public/private key pair

uid = userID

sub = subkey

Validity values:

r = revoked

d = disabled

e = expired

- = unknown

i = invalid

m = marginal

c = complete

Trust values:

q = undefined

- = unknown

n = never

m = marginal

c = complete

u = ultimate

Key algorithm values:

-1 = invalid

1 = RSA

2 = RSA Encrypt Only

3 = RSA Sign Only

16 = ElGamal (Diffie-Hellman)

17 = DSA

WITH-DELIMITER

The **WITH-DELIMITER** variable controls how the output of a key view (-kv) is formatted. The delimiter can be set to any character you want. For example, you might choose to set delimiter equal to a semi-colon (;).

Default Value

WITH-DELIMITER = ""

Notes

If you set **WITH-DELIMITER** to equal a semi-colon (;), then the output of your key view displays in the following format:

```
[pub]sec];validity;key_size;key_algorithm;keyid;creation_date;  
expiration_date;trust;username;
```

For a listing of abbreviation descriptions of validity, trust, and key_algorithm values, see the section above, "[WITH-COLONS](#)".

The tables in this appendix identify PGP's exit and error codes.

General errors

Error	Description
0	Exit OK, no error
1	invalid file
2	file not found
3	unknown file
4	batchmode error
5	bad argument
6	process interrupted
7	out of memory error

Keyring errors

Error	Description
10	key generation error
11	non-existing key error
12	keyring add error
13	keyring extract error
14	keyring edit error
15	keyring view error
16	keyring removal error
17	keyring check error
18	key signature error or key signature revoke error
19	key signature removal error

Encode errors

Error	Description
20	signature error
21	public key encryption error
22	encryption error
23	compression error

Decode errors

Error	Description
30	signature check error
31	public key decryption error
32	decryption error
33	decompression error

A quick reference of PGP options

The various command options are listed in the following tables.

Key options

The **-k** (key) option lists all the key management functions available in PGP.

-k is also used in combination with other options. The following table lists and describes these combinations.

Option	Description
-k (key help)	Display help on key options
-ka (key add)	Adds keys to the keyring
-kc (key check)	Check signatures
-ke (key edit)	Edit user ID, passphrase, trust options, default signing key, or add a designated revoker for your secret key.
-kd (key disable)	Revoke or disable keys on the keyring
-kds (key disable signatures)	Revoke signatures attached to keys on the keyring
-kg (key generate)	Generates a key
-kj (key join)	Reconstitute a split key locally.
-kl (key split)	Create a split key.
-kq (key join over a network)	Reconstitute a split key remotely, or over a network.
-kr (key remove)	Remove keys from the keyring or key server
-krs (key remove signatures)	Remove signatures attached to keys on the keyring

Option	Description
-ks (key sign)	Sign keys on the keyring
-ksx (key sign with expiration)	Sign keys on the keyring and add an expiration date to your signature.
-kv (key view)	View the keys on a keyring
-kvc (view fingerprint)	View the fingerprints of a set of keys
-kvv (view key and signatures)	View keys and signatures on the keyring
-kx (key extract)	Extract keys from the keyring so you can exchange it with others.
-kxa (key extract in ASCII-armored format)	Extract keys from the keyring in ASCII-armored format, which makes it easy to paste into email.

Email and file options

The following table identifies and describes PGP's command line options used to encrypt, decrypt, and manage files.

Option	Description
-a (ASCII)	When used with other options such as encryption or signing, converts a file to ASCII-armored format (creates a .asc file).
-c (conventionally encrypt)	Encrypt using conventional encryption.
-e (encrypt)	Encrypt using public key encryption.
-ea (encrypt ASCII)	Encrypts into ASCII text, which is suitable for sending through email channels.
-eat (encrypt text file in ASCII)	Encrypts a text file into a format suitable for sending through email channels.
-es (encrypt and sign)	Sign with your secret key and encrypt with the recipient's public key in one operation.
-et (encrypt text)	Specify that the plaintext be treated as ASCII text and converted to canonical text before encryption.
-ew (encrypt and wipe)	Indicates you want to automatically overwrite and delete the original plaintext file preventing someone from recovering it.
-f (filter)	Uses UNIX-style filter mode to read from standard input and write to standard output.
-feast (filter, encrypt, ascii, sign, text)	Decrypt a message, read from standard input and write to standard output.
-g (group)	Displays help on group options. See table below for -g combinations.
-h (help)	Displays summary of commands
-i (input)	Specifies a file containing all the input required by PGP when used in conjunction with full batch mode.

Option	Description
-k (key)	Displays help on key options. See table below for -k combinations.
-m (more)	Displays plaintext output on your screen.
-o (change output filename)	When used with other options such as encryption, decryption, checking signatures, and filter mode, specifies the output filename.
-p (preserve plaintext filename)	Recovers the original plaintext filename.
-s (sign)	Signs.
-sb (sign and break)	Creates a separate, detached signature certificate file.
-sem (sign, encrypt for eyes-only)	Specifies that the recipient's decrypted plaintext be shown only on the recipient's screen and not saved to disk.
-sta (sign plaintext ASCII text file)	Produces a clear-signed message suitable for distribution through email channels.
-t (text file)	Identifies the input file as a text file.
-u (user)	Identifies the key to use for signing.
-w (wipe)	Instructs PGP to wipe the file.
-z (passphrase)	Indicates that the text string immediately following the option is your passphrase

Group options

The **-g** option is always used in combination with another option. The following table lists these combinations and describes how they are used.

Option	Description
-g (group)	Displays help on group options.
-ga (group add)	Adds recipients or groups to a group.
-gr (group remove)	Removes recipients or groups from a group.
-gv (group view)	View a group.
-gvv (group view and content)	View a group and the recipients or groups it contains. Default is view all groups and their constituent keys or groups.

Common PGP functions

The following tables display command syntax for common PGP functions including:

- Creating, exchanging, and editing keys
- Encrypting information
- Decrypting information
- Signing files and verifying signatures
- Working with groups
- Working with ASCII and binary data
- Getting help

Key commands

What do you want to do?	Syntax
Create a key pair	pgp -kg
View all the keys on a keyring	pgp -kv <keyring_filename>
View a specific key on a keyring	pgp -kv [<userID>] { [<keyring_filename>] or [<keyserver_URL>] }
Display all certifying signatures attached to each key	pgp -kvv [<userID>] [<keyring>]
Check the signatures on a key	pgp -kc [<your_userID>] [<keyring>]
Display the fingerprint of a public key	pgp -kvc [<userID>] [<keyring>]
Get a key from a key server and put the key on your keyring	Requires two commands: pgp -kx <userID> <key_filename> <keyserver_URL> and pgp -ka <key_filename>
Extract your key to a file	pgp -kx <userID> <key_filename> [<keyring>]
Extract your key in ASCII-armored format	pgp -kxa <userID> <key_filename> [<keyring>]
Add a key to your keyring	pgp -ka <key_filename> [<keyring_name>]
Remove a key or a user ID from your keyring	pgp -kr <userID> <keyID> [<keyring_name>]
Add your key to a key server	pgp -kx <userID> <keyserver_URL>

What do you want to do?	Syntax
Change your default userID, passphrase, trust options, or add a designated revoker to a key	pgp -ke <userID> [<keyring>]
Revoke your key	pgp -kd <userID_of_key_being_revoked>
Disable or re-enable a key	pgp -kd <userID_of_key_being_disabled>
Create a split key	pgp -kl <userID_of_key_to_split>
Join a split key locally	pgp -kj <userID_of_key_to_join>
Join a split key over the network	Requires two commands: pgp -kj <userID_of_key_to_join> then pgp -kq <IPaddress_of_system_with_split_key>
Delete a key from a key server	pgp -kr <userID_of_key_to_remove> <keyserver_URL>
Display all the keys in a specific keyring	pgp <keyring_filename>
Remove selected signatures from a user ID on a keyring	pgp -krs <userID> [<keyring>]
Sign and certify someone else's public key on your keyring	pgp -ks <recipient's userID> [-u <your userID>] [<keyring>]
Sign a key with an expiration date on your signature.	pgp -ksx <recipient's userID> [-u <your userID>] [<keyring>]

For more detailed instructions and examples on creating and distributing keys, see [Chapter 3, “Creating and Exchanging Keys”](#). To learn more about key management and editing keys, see [Chapter 6, “Key Management Operations”](#).

Encryption commands

What do you want to do?	Syntax
Encrypt with conventional encryption	pgp -c <i><plaintext_filename></i>
Encrypt with public key encryption	pgp -e <i><plaintext_filename></i> <i><recipients_userID></i>
Encrypt into ASCII text	pgp -ea <i><plaintext_filename></i> <i><recipients_userID></i>
Encrypt a text file into ASCII text	pgp -eat <i><plaintext_filename></i> <i><recipients_userID></i>
Encrypt and filter a file to an application that reads STDIN	pgp -ef <i><plaintext_filename></i> <i><recipients_userID></i>
Encrypt to multiple recipients	pgp -e <i><plaintext_filename></i> <i><userID1></i> <i><userID2></i> <i><userID3></i> ...
Encrypt to a group	pgp -e <i><plaintext_filename></i> <i><group_name></i>
Encrypt for viewing by recipient only	pgp -sem <i><message.txt></i> <i><recipients_userID></i>
Encrypt and wipe original plaintext file	pgp -ew <i><message.txt></i> <i><recipients_userID></i>
Encrypt with an Additional Decryption Key (ADK)	pgp +ADKKEY=<userID> or pgp +ADKKEY=<keyID>
Enforce encrypting to an ADK	pgp +ENFORCEADK=on

For more detailed instructions and examples on encryption operations, see [“Encrypting information” on page 43](#).

Decryption commands

What do you want to do?	Syntax
Decrypt a message	pgp < <i>ciphertext_filename</i> > [-o < <i>plaintext_filename</i> >]
Decrypt an ASCII-armored message	pgp < <i>ASCII-armored_message.asc</i> >
Decrypt a message and view plaintext output on your screen	pgp -m < <i>ciphertext_filename</i> >
Decrypt and specify a plaintext filename for the output	pgp -o < <i>ciphertext_filename</i> >
Decrypt a message and recover the original plaintext filename	pgp -p < <i>ciphertext_filename</i> >
Decrypt a message, read from standard input and write to standard output	pgp -feast < <i>recipients_userID</i> > << < <i>input_filename</i> > > < <i>output_filename</i> >

For more detailed instructions and examples on decryption operations, see [“Decrypting information” on page 47](#).

Signing commands

What do you want to do?	Syntax
Sign a file	pgp -s <i><plaintext_filename></i> [-u <i><your_userID></i>]
Produce a clear-signed message	pgp -sat <i><plaintext_filename></i>
Sign and encrypt a file	pgp -es <i><plaintext_filename></i> <i><recipient's_userID></i> [-u <i><your_userID></i>]
Sign a plaintext ASCII text file	pgp -sta <i><plaintext_filename></i> [-u <i><your_userID></i>]
Create a detached signature	pgp -sb <i><plaintext_filename></i> [-u <i><your_userID></i>]
Sign and certify someone else's public key on your public keyring	pgp -ks <i><recipient's_userID></i> [-u <i><your_userID></i>] [<i><keyring></i>]
Verify a signature on a file	pgp <i><ciphertext_filename></i> [-o <i><plaintext_filename></i>]

For more detailed instructions and examples on signing files and verifying the authenticity of signatures, see [Chapter 5, “Working with Digital Signatures and Validation”](#).

Group commands

What do you want to do?	Syntax
Create a group	pgp -ga <i><group_name></i>
Add recipients to a group	pgp -ga <i><group_name></i> <i><userID_to_add></i>
Remove recipients from a group	pgp -gr <i><group_name></i> <i><userID_to_remove></i>
Display name and description of a group	pgp -gv <i><group_name></i>
Display the contents of a group	pgp -gvv <i><group_name></i>

For more detailed instructions and examples on working with groups, see [“Working with groups” on page 80](#).

ASCII and binary data commands

What do you want to do?	Syntax
Create a ciphertext file in ASCII-armored-64 format	pgp -sea <i><plaintext_filename></i> <i><recipient's_userID></i> or pgp -kxa <i><userID></i> <i><keyfile></i> [<i><keyring></i>]
Create a plaintext ASCII file	pgp -seat <i><message.txt></i> <i><recipient's_userID></i>
Send binary data files in ASCII-armored format	pgp -a <i><binary_filename></i>

For more detailed instructions and examples on working with ASCII and binary data, see [“Working with ASCII and binary data” on page 76](#).

Help commands

What do you want to do?	Syntax
Display a quick command usage summary for PGP	pgp -h
Display help for key options	pgp -k
Display help for group options	pgp -g

Attaching a Regular Expression to a Signature



This appendix describes the purpose of attaching a regular expression to a signature, lists the special characters used in a regular expression, and defines the regular expression syntax used in PGP.

Attaching a regular expression to a signature

The purpose of a regular expression on a signature is to restrict the scope of the target key's signature power. For example, a corporate administrator might place a signature with an attached regular expression on a sub-administrator, who controls the HR department, that states that he/she can only sign keys from "hr.nai.com".

The following special characters can be used in a regular expression:

- a pipe (|)
- parenthesis ()
- an asterisk (*)
- a plus sign (+)
- a question mark (?)
- brackets []
- a period (.)
- a caret (^)
- a dollar sign (\$)

When using any of these characters in a regular expression, put a backslash (\) in front of a literal character to distinguish it from one of the special characters.

For example, the following regular expression matches any email address from pgp.com, such as <joe@pgp.com> or <philz@pgp.com>.

`<.*@pgp\.com>`

Definitions of the regular expression syntax used in PGP

- A *regular expression* is zero or more branches separated by a pipe (`|`). The regular expression matches anything that matches one of the branches.
- A *branch* is zero or more pieces, concatenated. The branch looks for a match for the first piece, then looks for a match for the second piece, etc.
- A *piece* is an atom possibly followed by one of these characters: an asterisk (`*`), a plus sign (`+`), or a question mark (`?`).
 - An atom followed by an asterisk (`*`) matches a sequence of 0 or more matches of the atom.
 - An atom followed by a plus sign (`+`) matches a sequence of 1 or more matches of the atom.
 - An atom followed by a question mark (`?`) matches a match of the atom or the null string.
- An *atom* can be a regular expression in parentheses (`()`), a single character, or one of the following special characters. The atom matches a part of the userID as a sub expression within the larger regular expression, assuming that the beginning of the regular expression has already matched.
 - A period (`.`) represents matching any single character.
 - A caret (`^`) represents matching the null string at the beginning of the input string.
 - A dollar sign (`$`) represents matching the null string at the end of the input string.
 - A backslash (`\`) followed by a single character represents matching that character. A backslash (`\`) followed by a single character with no other significance represents matching that character.
- A *range* is a sequence of characters enclosed in brackets [`]`. The range normally matches any single character from the sequence.
 - If the sequence begins with a caret (`^`), it matches any single character not from the rest of the sequence.
 - Two characters in a sequence separated by a dash (`-`) represents the full list of ASCII characters between them. For example, **[0-9]** matches any decimal digit.
 - To include a literal end bracket (`]`) in the sequence, make it the first character (following a possible `^`). To include a literal dash (`-`) in the sequence, make it the first or last character.

Index

Symbols

.asc files

setting PGP to produce [85](#), [109](#)

A

-a option [109](#)

adding

a designated revoker [62](#)

a key to a key server [112](#)

an expiration date to a signature [55](#)

an expiration date to your signature [113](#)

hours to system time to get GMT [100](#)

keys

to key servers [42](#)

to your keyring [41](#)

private keys to a keyring [107](#), [112](#)

public keys to a keyring [107](#), [112](#)

recipients or groups to a group [111](#)

Additional Decryption Keys

adding to all generated keys [114](#)

an overview of [69](#)

appropriate use [70](#)

definition of [69](#)

encrypting to [84](#)

implementing [71](#)

incoming ADKs [70](#)

key policy [71](#)

outgoing ADKs [70](#)

protecting [71](#)

security [71](#)

ADKKEY parameter [84](#)

ADKs

See Additional Decryption Keys

ARMOR parameter [85](#)

ASCII text file

sending to a different machine environment [77](#)

signing with your secret key [51](#)

ASCII-armored format [77](#), [85](#)

converting a file into [77](#)

creating files in [109](#)

decrypting messages in [77](#)

encrypting to [109](#)

sending a public key in [77](#)

sending binary data files in [117](#)

sending binary data in [77](#)

atom, definition [120](#)

attaching a regular expression to a signature [119](#)

B

backwards compatibility

enabling compatibility with v2.6.2 [26](#), [88](#)

BATCHMODE flag

set to eliminate unnecessary questions [74](#)

BATCHMODE parameter [86](#)

binary data

encrypting and transmitting [76](#)

binary data files

sending in ASCII-armored format [77](#), [117](#)

branch, definition [120](#)

C

-c option [43](#), [109](#)

canonical text

converting to before encrypting [99](#)

CERT_DEPTH parameter [86](#)

- certificate servers
 - See key servers
- certifying
 - public keys [20](#)
- changing your passphrase [60](#)
- checking
 - a key's validity [53](#)
 - signatures on a key [107](#)
- choosing
 - a key type [29](#)
- CIPHERNUM parameter [87](#)
- CLEARSIG parameter [87](#)
- clear-signed message
 - an example of [50](#)
 - creating [50](#)
- clear-signing
 - producing human-readable signatures [87](#)
- command line
 - setting configuration parameters on [84](#)
- command line options, description of
 - a [109](#)
 - c [109](#)
 - e [109](#)
 - ea [109](#)
 - eat [109](#)
 - es [109](#)
 - et [109](#)
 - ew [109](#)
 - f [109](#)
 - feast [109](#)
 - g [109](#), [111](#)
 - ga [111](#)
 - gr [111](#)
 - gv [111](#)
 - gvv [111](#)
 - h [109](#)
 - i [109](#)
 - k [107](#), [110](#)
 - ka [107](#)
 - kc [107](#)
 - kd [107](#)
 - kds [107](#)
 - ke [107](#)
 - kg [107](#)
 - kj [107](#)
 - kl [107](#)
 - kq [107](#)
 - kr [107](#)
 - krs [107](#)
 - ks [108](#)
 - ksx [108](#)
 - kv [108](#)
 - kvc [108](#)
 - kvv [108](#)
 - kx [108](#)
 - kxa [108](#)
 - m [110](#)
 - o [110](#)
 - p [110](#)
 - s [110](#)
 - sb [110](#)
 - sem [110](#)
 - sta [110](#)
 - t [110](#)
 - u [110](#)
 - w [110](#)
 - z [110](#)
- commands
 - display a summary of [27](#)
- comment header
 - specifying text of [88](#)
- COMMENT parameter [88](#)
- compatibility
 - with PGP 2.6.2 [26](#), [88](#)
- COMPATIBLE parameter [26](#), [88](#)
- COMPLETES_NEEDED parameter [89](#)
- COMPRESS parameter [89](#)
- compression
 - before encryption, setting [89](#)

- configuration file
 - description of [83](#)
 - learning about [83](#)
 - locating
 - on UNIX [24](#)
 - on Windows [25](#)
 - specifying location of [23](#)
- configuration parameters
 - ADDKEY [84](#)
 - ARMOR [85](#)
 - BATCHMODE [86](#)
 - CERT_DEPTH [86](#)
 - CIPHERNUM [87](#)
 - CLEARSIG [87](#)
 - COMMENT [88](#)
 - COMPATIBLE [26](#), [88](#)
 - COMPLETES_NEEDED [89](#)
 - COMPRESS [89](#)
 - ENCRYPTTOSELF [89](#)
 - ENFORCEADK [90](#)
 - entering as long options [26](#)
 - FASTKEYGEN [90](#)
 - FORCE [91](#)
 - GROUPSFILE [91](#)
 - HASHNUM [92](#)
 - INTERACTIVE [92](#)
 - MARGINALS_NEEDED [92](#)
 - MYNAME [93](#)
 - PAGER [93](#)
 - PASSPHRASE-FD [94](#)
 - PGP_MIME [94](#)
 - PGP_MIMEPARSE [95](#)
 - RANDOMDEVICE [96](#)
 - RANDSEED [96](#)
 - RSAVER [97](#)
 - SECRING [97](#)
 - setting on the command line [84](#)
 - SHOWPASS [98](#)
 - SIGTYPE [98](#)
 - STATUS-FD [99](#)
 - TEXTMODE [99](#)
 - TMP [99](#)
 - TZFIX [100](#)
 - VERBOSE [101](#)
 - WITH-COLONS [101](#)
 - WITH-DELIMITER [103](#)
- configuration values
 - specifying [83](#)
- configuring
 - ASCII-armored format [85](#)
 - comment header [88](#)
 - compression [89](#)
 - number of completely trusted introducers needed [89](#)
 - PGP [19](#)
 - signature format [87](#)
 - use of Additional Decryption Keys [84](#)
- confirmation questions
 - eliminating [74](#)
- confirming
 - multiple key adds [92](#)
- conventional encryption
 - encrypting with [43](#)
 - setting PGP to encrypt with [109](#)
- converting files
 - into ASCII-armored format [77](#)
- copying
 - a key from a server [112](#)
 - keys [112](#)
- creating [65](#)
 - a clear-signed message [116](#)
 - a detached signature [51](#)
 - Additional Decryption Keys [71](#)
 - ciphertext files in ASCII-armored format [117](#)
 - detached signatures [51](#)
 - key pairs [30](#), [112](#)
 - plaintext ASCII files [117](#)
 - signature certificates detached from the document [116](#)
 - split keys [65](#)
- Customer service, contacting [x](#)

D

data recovery
 additional decryption keys [69](#)
 versus key recovery [70](#)

decrypt
 command syntax [47](#)

decrypted plaintext
 display only on recipients screen [46](#)
 viewing one screen at a time [93](#)

decrypting
 and recovering original filename [115](#)
 and renaming the plaintext filename
 output [48](#)
 and specifying a filename for the output
 [115](#)
 and viewing plaintext output [115](#)
 on your screen without writing to a file
 [47](#)
 ASCII-armored messages [77](#), [115](#)
 email [21](#)
 messages [115](#)
 read from standard input [115](#)
 writing to standard output [115](#)
 with a passphrase [47](#)
 with your secret key [47](#)

decryption
 an overview of [47](#)

default key ID
 for signatures, setting [93](#)

default user ID
 setting [61](#)

deleting
 keys from a server [72](#)
 recipients from groups [111](#)

designate someone as a trusted introducer [54](#)

designated revoker
 to your key, adding [62](#)

destroying a plaintext file [78](#)

detached signature
 creating [51](#)

Diffie-Hellman/DSS keys
 an overview of [29](#)

digital signature
 verifying [52](#)

directory pathname
 specifying for temporary files [99](#)

disabling
 keys [65](#), [113](#)
 on your keyring [107](#)

displaying
 all keys in a specific keyring [113](#)
 all signatures attached to a key [112](#)
 contents of your public keyring [112](#)
 plaintext output on your screen [110](#)
 the fingerprint of a public key [112](#)

distributing
 public keys [19](#), [40](#)

E

-e option [44](#), [109](#)

-ea option [109](#)

-eat option [109](#)

echo passphrase to user [98](#)

edit
 a set of keys [107](#)
 the trust parameters for a public key [54](#)
 your default user ID [61](#)
 your key [58](#)
 your passphrase [60](#)
 your user ID [59](#)

editing
 your key [113](#)

eliminate interaction
 using FORCE [91](#)

eliminating
 confirmation questions with FORCE [74](#)
 unnecessary questions with
 BATCHMODE [74](#)

eliminating confirmation questions [74](#)

- email
 - decrypting 21
 - encrypting 20
 - signing 20
 - verifying 21
 - enabling
 - disabled keys 113
 - encrypted information
 - exchanging 43
 - encrypting
 - a plaintext file 50
 - a text file 114
 - and filtering to an application that reads standard input 114
 - and signing in one operation 50
 - and wiping the original plaintext file 78, 114
 - binary data 76
 - email messages 20
 - for any number of recipients 45, 114
 - for viewing by recipient only 46, 114
 - into ASCII text 114
 - to
 - a passphrase 43
 - an ADK, forcefully 90
 - groups 46
 - multiple recipients 45
 - your own key automatically 46, 89
 - to a group 114
 - with
 - an Additional Decryption Key 114
 - conventional encryption 43, 109, 114
 - public key encryption 44, 109
 - the -a option 109
 - the ADKKEY parameter 84
 - the recipient's public key 114
 - encryption options
 - c 109
 - e 109
 - f 109
 - g 109, 111
 - ENCRYPTTOSELF parameter 46, 89
 - enforce encrypting to an ADK 90
 - ENFORCEADK parameter 90
 - entering configuration parameters
 - on the command line 26
 - entropy
 - acquiring 96
 - entropy pool
 - identifying 96
 - environment variables
 - PGPPASS 79
 - PGPPASSFD 80
 - PGPPATH 24
 - error codes 105
 - es option 109
 - et option 109
 - ew option 109
 - exchanging
 - encrypted information 43
 - public keys 19
 - exit status codes 75, 105
 - expiration date
 - adding to your signature 113
 - exportable signatures 98
 - exporting
 - your key to a file 40
 - extracting keys
 - from a keyring 108
 - from key servers 42
 - to a file 40, 112
- ## F
- f option 109
 - fast key generation 90
 - FASTKEYGEN parameter 90
 - feast option 109
 - files
 - wiping 110
 - filtering 75
 - fingerprints
 - displaying 112

FORCE parameter 91

using to eliminate interaction with PGP
74

G

-g option 109, 111

-ga option 81, 111

generating

key pairs 30

keys 107

-gr option 81, 111

Greenwich Mean Time

adjusting system time to get 100

groups

adding recipients to 81, 111, 117

creating 81, 117

displaying contents of 117

displaying help for 109, 111

displaying name and description of 117

encrypting to 46

removing recipients from 81, 111, 117

viewing 81, 111

viewing the group and its keys 111

groups file

specifying location of 23

specifying with GROUPSFILE 91

GROUPSFILE parameter 91

-gv option 111

-gvv option 111

H

-h option 27, 109

HASHNUM parameter 92

help

displaying command summary 109

getting 109, 111

on group options 109, 111

on key options 107, 110

using -h 109

I

-i option 109

INTERACTIVE parameter 92

introduction to

PGP Command Line 17

K

-k option 107, 110

-ka option 107

-kc option 107

-kd option 107, 113

-kds option 107

-ke option 107

key extraction 40

key ID

specifying the default with MYNAME
93

key pairs

creating 19, 30

description of 30

generating 30

making 30

key servers

adding keys to 42, 112

copying keys from 112

deleting keys from 72, 113

retrieving keys from 42

key shares 65

key types

an overview of 29

choosing the right one 29

key viewing 39

- keyrings
 - adding keys to [41](#), [107](#), [112](#)
 - checking signatures on [63](#)
 - confirming multiple key additions [92](#)
 - displaying contents of [112](#)
 - locating
 - on UNIX [24](#)
 - on Windows [25](#)
 - managing [57](#)
 - overview of [17](#)
 - removing
 - keys from [41](#), [58](#)
 - user IDs from [58](#)
 - specifying location of files [23](#)
 - verifying the contents of [63](#)
 - viewing all the keys on [57](#)
 - keys
 - adding
 - a designated revoker [62](#)
 - to a key server [42](#)
 - to your keyring [41](#)
 - copying [112](#)
 - from the server [112](#)
 - creating ADKs [71](#)
 - disabling [65](#), [113](#)
 - displaying
 - a key's fingerprint [112](#)
 - a key's signatures [112](#)
 - all keys on a keyring [113](#)
 - signatures on [53](#)
 - distributing [40](#)
 - editing [58](#)
 - by adding a designated revoker [62](#)
 - trust settings [54](#)
 - your default user ID [61](#)
 - your passphrase [60](#)
 - your user ID [59](#)
 - enabling [113](#)
 - encrypt to your own automatically [46](#)
 - exchanging
 - with others [40](#)
 - extracting to a file [40](#)
 - generating [30](#)
 - granting trust for [54](#)
 - making available to others [42](#)
 - overview of [17](#)
 - rejoining split keys [65](#)
 - removing
 - from a key server [72](#)
 - from your keyring [41](#)
 - removing signatures from [55](#)
 - retrieving
 - from key servers [42](#)
 - revoking [64](#), [113](#)
 - signing [54](#)
 - signing with an expiration date on your signature [55](#)
 - splitting [65](#)
 - verifying
 - signature on [39](#)
 - viewing
 - all in a group [111](#)
 - fingerprint on [39](#), [53](#)
 - on your keyring [57](#)
 - signatures associated with it [39](#)
 - kg option [107](#), [112](#)
 - kj option [107](#), [113](#)
 - kl option [107](#), [113](#)
 - kq option [107](#), [113](#)
 - kr option [107](#)
 - krs option [55](#), [107](#)
 - ks option [108](#)
 - ksx option [55](#), [108](#)
 - kv option [108](#)
 - kvc option [108](#)
 - kvv option [108](#)
 - kx option [108](#)
 - kxa option [108](#)
- L**
- learning about
 - PGP configuration file [83](#)
 - level of trust
 - displaying [53](#)

listing keys on your keyring [57](#)

locating

configuration file [23](#)

groups file [23](#)

keyring files [23](#)

PGP files [23](#)

randseed file [23](#)

M

-m option [110](#)

making

key pairs [30](#)

managing your keyring [57](#)

marginally trusted introducers

identifying the minimum [92](#)

MARGINALS_NEEDED parameter [92](#)

messages

decrypting [115](#)

meta signatures [98](#)

meta-introducers [53](#)

and trust [53](#)

MSDOS

running PGP in batch mode from [73](#)

multiple recipients

encrypting to [45](#)

MYNAME parameter [93](#)

N

Network Associates

contacting

Customer Service [x](#)

training [xii](#)

non-exportable signatures [98](#)

O

-o option [110](#)

output

view using PAGER [93](#)

overview

of Additional Decryption Keys [69](#)

of Diffie-Hellman keys [29](#)

of key concepts [17](#)

of keyrings [17](#)

of private keys [17](#)

of RSA keys [29](#)

of RSA Legacy keys [29](#)

P

-p option [110](#)

PAGER parameter [93](#)

passing your passphrase from another application [80](#)

PASSPHRASE-FD parameter [94](#)

passphrases

alternative ways to work with [79](#)

changing [60](#)

creating [32](#)

encrypting to [43](#)

inability to retrieve [32](#)

passing from another application [80](#)

seeing as you type [98](#)

specify file descriptor

using PASSPHRASE-FD [94](#)

supplying to PGP using PGPPASSFD [80](#)

using the -z option [110](#)

PGP

exit status codes [75](#)

starting [25](#)

PGP Command Line

introduction to [17](#)

PGP command syntax

general guidelines [26](#)

PGP configuration file

description of [83](#)

learning about [83](#)

specifying location of [23](#)

PGP files

specifying location of [23](#)

- PGP groups file
 - specifying location of [23](#)
- PGP Key Wizard
 - using to create key pairs [30](#)
- PGP Security
 - contacting in the U.S. [xi](#)
- PGP Version 2.6.2
 - enabling compatibility with [88](#)
 - setting compatibility with [26](#)
- pgp.cfg
 - description of [83](#)
 - specifying location of [23](#)
- PGP_MIME parameter [94](#)
- PGP_MIMEPARSE parameter [95](#)
- pgpgroup.pgr
 - specifying location of [23](#)
- PGPPASS environment variable
 - storing your passphrase with [79](#)
- PGPPASSFD environment variable
 - supplying your passphrase with [80](#)
- PGPPATH
 - identify the location of PGP configuration file using [24](#)
- piece, definition [120](#)
- plaintext
 - converting to canonical text [99](#)
 - wiping the original plaintext file [114](#)
- plaintext file
 - wiping [78](#)
- plaintext filename
 - recovering [48](#)
- policy
 - for ADKs [71](#)
- preserving the original plaintext filename [48](#)
- private and public key pairs
 - creating [19](#)
- private keyring
 - locating
 - on UNIX [24](#)
 - on Windows [25](#)
 - specifying filename and path [97](#)
- private keys
 - adding to a keyring [107](#), [112](#)
 - creating [19](#)
 - key pairs [19](#)
 - overview [17](#)
 - signing with [50](#)
- producing
 - a clear-signed message [49](#)
- protecting
 - Additional Decryption Keys [71](#)
- public key encryption [44](#)
 - specifying that PGP use [109](#)
- public keyring
 - checking signatures on [112](#)
 - displaying contents of [112](#)
 - locating
 - on UNIX [24](#)
 - on Windows [25](#)
 - specify filename and path [95](#)
 - verifying the contents [63](#)
- public keys
 - adding
 - to key servers [42](#)
 - to your keyring [41](#)
 - adding to a keyring [107](#), [112](#)
 - certifying [20](#)
 - creating [19](#)
 - key pairs [19](#)
 - displaying the fingerprint [112](#)
 - distributing your [40](#)
 - exchanging with others [19](#), [40](#)
 - getting from key servers [42](#)
 - giving to other users [19](#)
 - making available to others [42](#)
 - removing from your keyring [41](#)
 - retrieving from a key server [112](#)
 - revoking [64](#)

- sending in ASCII-armored format 77
- trading with other users 19
- validating 20
- pubring.pkr
 - specifying location of 23

R

- random number seed file
 - locating
 - on UNIX 24
 - on Windows 25
 - specify filename and path 96
 - specifying location of 23
- RANDOMDEVICE parameter 96
- RANDSEED parameter 96
- randseed.rnd
 - specifying location of 23
- range, definition 120
- reconstituting
 - a split key 67
 - locally 67
 - over the network 67
- recovering
 - the original plaintext filename 48
- redirecting status messages
 - to a file descriptor 99
- re-enable a key 113
- regular expression, definition 120
- regular expressions
 - attaching to a signature 119
 - definitions of syntax used in PGP 120
 - list of special characters 119
- rejoining
 - a split key 67
 - locally 67
 - over the network 67
 - keys 65

- removing
 - keys
 - from keyrings 41, 58, 107, 112
 - recipients or groups from a group 111
 - signatures
 - attached to keys on your keyring 107
 - from a userID on a keyring 113
 - user IDs
 - from keyrings 58, 112
- revoking
 - a key using a designated revoker 62
 - keys 64, 113
 - on your keyring 107
 - signatures
 - attached to keys on the keyring 107
- RSA keys
 - an overview of 29
 - specifying which version to use 97
- RSA Legacy keys
 - an overview of 29
 - specifying PGP to use 97
- RSAVER parameter 97
- running PGP
 - in batch mode 73

S

- s option 110
- sb option 110
- scripts
 - using with PGP 73
- secret key
 - signing with 50
- secret keyring
 - specifying filename and path 97
- SECRING parameter 97
- secring.skr
 - specifying location of 23
- sem option 110

- sending
 - a public key
 - in ASCII-armored format 77
 - ASCII text files
 - to different machine environments 77
 - binary data files in ASCII-armored format 77, 117
- servers
 - copying keys from 112
- setting location of
 - configuration file 23
 - groups file 23
 - keyring files 23
 - PGP files 23
 - random number seed file 23
- several recipients
 - encrypting to 45
- SHOWPASS parameter 98
- signature certificates 51
- signatures
 - checking a public key's 112
 - displaying 112
 - producing human-readable 87
 - removing from a key 55
 - types
 - Export 98
 - Meta 98
 - Non 98
 - Trusted 98
 - verifying a detached one 52
 - verifying a digital one 52
 - viewing one's on a key 53
 - with an expiration date 55
- signed files
 - storing 52
- signing
 - a key 54
 - a key with an expiration date on your signature 113
 - a plaintext ASCII text file 51, 116
 - a plaintext file 49 to 50, 116
 - and encrypting 50, 116
 - command syntax 110
 - email messages 20
 - files without encrypting 52
 - keys on your keyring 108
 - to certify someone else's public key on your public keyring 113, 116
 - with a specific private key 50
- SIGTYPE parameter 98
- specifying
 - configuration values 83
 - key to sign with 50
 - location of
 - configuration file 23
 - groups file 23, 91
 - keyring files 23
 - PGP files 23
 - randseed file 23
- split key
 - creating 65
 - rejoining 67
- split keys
 - creating 113
 - joining locally 113
 - joining over the network 113
- splitting
 - keys 65
- sta option 110
- starting
 - PGP 25
- status file descriptor 99
- status messages
 - writing to an alternative output stream 99
- STATUS-FD parameter 99

storing

- signed files [52](#)
- your passphrase with PGPPASS [79](#)

summary of commands [27](#)

T

-t option [110](#)

technical support

- information needed from user [xii](#)
- online [xi](#)
- phone numbers for [xi](#)

temp files

- specifying directory for [99](#)

TEXTMODE parameter [99](#)

time

- adjusting to get GMT [100](#)

TMP parameter [99](#)

training for Network Associates products [xii](#)

- scheduling [xii](#)

transmitting

- binary data [76](#)
- your passphrase
 - from another application [80](#)

trust

- and meta-introducers [53](#)
- definition of [54](#)
- granting for key validation [54](#)
- setting number of completes needed [89](#)
- setting number of marginally trusted introducers needed [92](#)
- setting with CERT_DEPTH [86](#)

trusted introducers

- nesting level of trust [86](#)
- setting number of completes needed [89](#)

trusted signatures [98](#)

TZFIX parameter [100](#)

U

-u option [110](#)

UNIX shell script

- running PGP in batch mode from [73](#)

UNIX-style filter

- the -f option [109](#)
- using PGP as a [75](#)

user ID

- adding a new one [59](#)
- change your primary [59](#)
- setting your default [61](#)

V

validating

- public keys [20](#)

validity

- checking a key's [53](#)
- definition of [53](#)

VERBOSE parameter [101](#)

verifying

- a detached signature [52](#)
- a digital signature [52](#)
- a fingerprint [53](#)
- a public key
 - over the phone [53](#)
- email [21](#)
- signatures on a key [53](#)

viewing

- a key's fingerprint [53](#)
- decrypted plaintext output
 - one screen at a time [93](#)
- decrypted plaintext output on your screen [47](#)
- groups [111](#)
- groups and their keys [111](#)
- keys and signatures on a keyring [108](#)
- keys on a keyring [108](#)
- signatures on a key [53](#)
- the fingerprints of a set of keys [108](#)

W

-w option [110](#)

wiping

files [110](#)

the original plaintext file [114](#)

using the -w option [110](#)

your disk [78](#)

WITH-COLONS parameter [101](#)

WITH-DELIMITER parameter [103](#)

Z

-z option [80](#), [110](#)

zero exit status code [75](#)

