PGP User's Guide, Volume II: Special Topics

Philip Zimmermann

11 October 94

1 Introduction

Phil's Pretty Good Software presents PGP(tm) Pretty Good(tm) Privacy. Public key encryption for the masses. Document revised 11 October 94 for PGP version 2.6.2 – 11 Oct 94. Software by Philip Zimmermann, and many others.

Synopsis: PGP(tm) uses public-key encryption to protect E-mail and data files. Communicate securely with people you've never met, with no secure channels needed for prior exchange of keys. PGP is well featured and fast, with sophisticated key management, digital signatures, data compression, and good ergonomic design.

Contents

2 Quick overview

Pretty Good(tm) Privacy (PGP), from Phil's Pretty Good Software, is a high security cryptographic software application for MSDOS, Unix, VAX/VMS, and other computers. PGP combines the convenience of the Rivest-Shamir-Adleman (RSA) public key cryptosystem with the speed of conventional cryptography, message digests for digital signatures, data compression before encryption, good ergonomic design, and sophisticated key management.

This volume II of the PGP User's Guide covers advanced topics about PGP that were not covered in the "PGP User's Guide, Volume I: Essential Topics". You should first read the Essential Topics volume, or this manual won't make much sense to you. Reading this Special Topics volume is optional, except for the legal issues section, which everyone should read.

3 Special topics

3.1 Selecting keys via key ID

In all commands that let the user type a user ID or fragment of a user ID to select a key, the hexadecimal key ID may be used instead. Just use the key ID, with a prefix of `0x`, in place of the user ID. For example:

```
pgp -kv 0x67F7
```

This would display all keys that had 67F7 as part of 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 unambiguously pick which key you want by specifying the key ID.

3.2 Separating signatures from messages

Normally, signature certificates are physically attached to the text they sign. This makes it convenient in simple cases to check signatures. It is desirable in some circumstances to have signature certificates stored separately from the messages they sign. It is possible to generate signature certificates that are detached from the text they sign. To do this, combine the `b` (break) option with the `s` (sign) option. For example:

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

This example produces an isolated signature certificate in a file called *letter.sig*. The contents of letter.txt are not appended to the signature certificate.

After creating the signature certificate file (letter.sig in the above example), send it along with the original text file to the recipient. The recipient must have both files to check the signature integrity. When the recipient attempts to process the signature file, PGP notices that there is no text in the same file with the signature and prompts the user for the filename of the text. Only then can PGP properly check the signature integrity. If the recipient knows in advance that the signature is detached from the text file, she can specify both filenames on the command line:

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

or:

```
pgp letter letter.txt
```

PGP will not have to prompt for the text file name in this case.

A detached signature certificate is useful if you want to keep the signature certificate in a separate certificate log. A detached signature of an executable program is also useful for detecting a subsequent virus infection. It is also useful if more than one party must sign a document such as a legal contract, without nesting signatures. Each person's signature is independent.

If you receive a ciphertext file that has the signature certificate glued to the message, you can still pry the signature certificate away from the message during the decryption. You can do this with the -b option during decrypt, like so:

```
pgp -b letter
```

This decrypts the letter.pgp file and if there is a signature in it, PGP checks the signature and detaches it from the rest of the message, storing it in the file letter.sig.

3.3 Decrypting the message and leaving the signature on it

Usually, you want PGP to completely unravel a ciphertext file, decrypting it and checking the nested signature if there is one, peeling away the layers until you are left with only the original plaintext file.

But sometimes you want to decrypt an encrypted file, and leave the inner signature still attached, so that you are left with a decrypted signed message. This may be useful if you want to send a copy of a signed document to a third party, perhaps re-enciphering it. For example, suppose you get a message signed by Charlie, encrypted to you. You want to decrypt it, and, leaving Charlie's signature on it, you want to send it to Alice, perhaps re-enciphering it with Alice's public key. No problem. PGP can handle that.

To simply decrypt a message and leave the signature on it intact, type:

```
pgp -d letter
```

This decrypts letter.pgp, and if there is an inner signature, it is left intact with the decrypted plaintext in the output file.

Now you can archive it, or maybe re-encrypt it and send it to someone else.

3.4 Sending ASCII text files across different machine environments

You may use PGP to encrypt any kind of plaintext file, binary 8-bit data or ASCII text. Probably the most common usage of PGP will be for E-mail, when the plaintext is ASCII text.

ASCII text is sometimes 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. This is a sad fact of life.

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. For example, the popular KERMIT communication protocol can convert text to canonical form when transmitting it to another system. This gets converted back to local text line terminators by the receiving KERMIT. This makes it easy to share text files across different systems.

But encrypted text cannot be automatically converted by a communication protocol, because the plaintext is hidden by encipherment. To remedy this inconvenience, PGP lets you specify that the plaintext should be treated as ASCII text (not binary data) and should be converted to canonical text form before it gets encrypted. At the receiving end, the decrypted plaintext is automatically converted back to whatever text form is appropriate for the local environment.

To make PGP assume the plaintext is text that should be converted to canonical text before encryption, just add the `t` option when encrypting or signing a message, like so:

```
pgp -et message.txt her_userid
```

This mode is automatically turned off if PGP detects that the plaintext file contains what it thinks is non-text binary data.

If you need to use the `-t` option a lot, you can just turn on the TEXTMODE flag in the PGP configuration file. That's what I do.

For PGP users that use non-English 8-bit character sets, when PGP converts text to canonical form, it may convert data from the local character set into the LATIN1 (ISO 8859-1 Latin Alphabet 1) character set, depending on the setting of the CHARSET parameter in the PGP configuration file. LATIN1 is a superset of ASCII, with extra characters added for many European languages.

3.5 Using PGP as a better uuencode

A lot of people in the Unix world send binary data files through E-mail channels by using the Unix *uuencode* utility to convert the file into printable ASCII characters that can be sent via email. No encryption is involved, so neither the sender nor the recipient need any special keys. The uuencode format was designed for a similar purpose as PGP's radix-64 ASCII transport armor format described in the "Sending Ciphertext Through E-mail Channels: Radix-64 Format" section, but not as good. A different radix-64 character set is used. Uuencode has its problems, such as 1) several slightly incompatible character sets for different versions of uuencode in the MSDOS and Unix worlds, and 2) the data can be corrupted by some E-mail gateways that strip trailing blanks or do other modifications to the character set used by uuencode.

PGP may be used in a manner that offers the same general features as uuencode, and then some. You can get PGP to just convert a file into PGP's radix-64 ASCII transport armor format, but you don't have to encrypt the file or sign it, so no keys are needed by either party. Simply use the `-a` option alone. For example:

```
pgp -a filename
```

This would produce a radix-64 armored file called *filename.asc.*

If you read the "Sending Ciphertext Through E-mail Channels: Radix-64 Format" section, you will see that PGP's approach offers several important advantages over the uuencode approach:

- PGP will break big files up into chunks small enough to E-mail.
- PGP will append a CRC error detection code to each chunk.
- PGP will attempt to compress the data before converting it to radix-64 armor.
- PGP's radix-64 character set is more resilient to E-mail character conversions than the one used by uuencode.
- Textfiles can be converted by the sender to canonical text format, as explained in the "Sending ASCII Text Files Across Different Machine Environments" section.

The recipient can restore the sender's original filename by unwrapping the message with PGP's `-p` option. You can use `pgp -a` in any situation in which you could have used uuencode, if the recipient also has PGP. PGP is a better uuencode than uuencode.

3.6 Leaving no traces of plaintext on the disk

After PGP makes a ciphertext file for you, you can have PGP automatically overwrite the plaintext file and delete it, leaving no trace of plaintext on the disk so that no one can recover it later using a disk block scanning utility. This is useful if the plaintext file contains sensitive information that you don't want to keep around.

To wipe out the plaintext file after producing the ciphertext file, just add the `w` (wipe) option when encrypting or signing a message, like so:

```
pgp -esw message.txt her_userid
```

This example creates the ciphertext file *message.pgp*, and the plaintext file *message.txt* is destroyed beyond recovery.

Obviously, you should be careful with this option. Also note that this 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. Also, it overwrites the file only once, which is enough to thwart conventional disk recovery efforts, but not enough to withstand a determined and sophisticated effort to recover the faint magnetic traces of the data using special disk recovery hardware.

3.7 Displaying decrypted plaintext on your screen

To view the decrypted plaintext output on your screen (like the Unix-style `more` command), without writing it to a file, use the `-m` (more) option while decrypting:

```
pgp -m ciphertextfile
```

This displays the decrypted plaintext display on your screen one screenful at a time.

3.8 Making a message for her eyes only

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

```
pgp -sem message.txt her_userid
```

Later, when the recipient decrypts the ciphertext with her secret key and pass phrase, the plaintext will be displayed on her screen but will not be saved to disk. The text will be displayed as it would if she used the Unix `more` command, one screenful at a time. If she wants to read the message again, she will have to decrypt the ciphertext again.

This feature is the safest way for you to prevent your sensitive message from being inadvertently left on the recipient's disk. This feature was added at the request of a user who wanted to send intimate messages to his lover, but was afraid she might accidentally leave the decrypted messages on her husband's computer.

Note that this feature will not prevent a clever and determined person from finding a way to save the decrypted plaintext to disk— it's to help prevent a casual user from doing it inadvertently.

3.9 Preserving the original plaintext filename

Normally, PGP names the decrypted plaintext output file with a name similar to the input ciphertext filename, but dropping the extension. Or, you can override that convention by specifying an output plaintext filename on the command line with the -o option. For most E-mail, this is a reasonable way to name the plaintext file, because you get to decide its name when you decipher it, and your typical E-mail messages often come from useless original plaintext filenames like *to_phil.txt*.

But when PGP encrypts a plaintext file, it always saves the original filename and attaches it to the plaintext before it compresses and encrypts the plaintext. Normally, this hidden original filename is discarded by PGP when it decrypts, but you can tell PGP you want to preserve the original plaintext filename and use it as the name of the decrypted plaintext output file. This is useful if PGP is used on files whose names are important to preserve.

To recover the original plaintext filename while decrypting, add the -p option, like so:

```
pgp -p ciphertextfile
```

I usually don't use this option, because if I did, about half of my incoming E-mail would decrypt to the same plaintext filenames of *to_phil.txt* or *prz.txt*.

3.10 Editing your user ID or pass phrase

Sometimes you may need to change your pass phrase, perhaps because someone looked over your shoulder while you typed it in.

Or you may need to change your user ID, because you got married and changed your name, or maybe you changed your E-mail address. Or maybe you want to add a second or third user ID to your key, because you may be known by more than one name or E-mail address or job title. PGP lets you attach more than one user ID to your key, any one of which may be used to look up your key on the key ring.

To edit your own userid or pass phrase for your secret key:

```
pgp -ke userid [keyring]
```

PGP prompts you for a new user ID or a new pass phrase.

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 will have to do that in a separate operation.

The optional [keyring] parameter, if specified, must be a public keyring, not a secret keyring. The userid field must be your own userid, which PGP knows is yours because it appears on both your public keyring and your secret keyring. Both keyrings will be updated, even though you only specified the public keyring.

The -ke command works differently depending on whether you use it on a public or secret key. It can also be used to edit the trust parameters for a public key.

3.11 Editing the trust parameters for a public key

Sometimes you need to alter the trust parameters for a public key on your public key ring. For a discussion on what these trust parameters mean, see the section "How Does PGP Keep Track of Which Keys are Valid?" in the Essential Topics volume of the PGP User's Guide.

To edit the trust parameters for a public key:

```
pgp -ke userid [keyring]
```

The optional [keyring] parameter, if specified, must be a public keyring, not a secret keyring.

3.12 Checking if everything is OK on your public key ring

Normally, PGP automatically checks any new keys or signatures on your public key ring 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 key ring. But perhaps you may want to explicitly force PGP to perform a comprehensive analysis of your public key ring, 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 key ring. To force PGP to perform a full analysis of your public key ring, use the -kc (key ring check) command:

```
pgp -kc
```

You can also make PGP check all the signatures for just a single selected public key by:

```
pgp -kc userid [keyring]
```

For further information on how the backup copy of your own key is checked, see the description of the BAKRING parameter in the configuration file section of this manual.

3.13 Verifying a public key over the phone

If you get a public key from someone that is not certified by anyone you trust, how can you tell if it's really their key? The best way to verify an uncertified key is to verify it over some independent channel other than the one you received the key through. One convenient way to tell, if you know this person and would recognize them on the phone, is to call them and verify their key over the telephone. Rather than reading their whole tiresome (ASCII-armored) key to them over the phone, you can just read their key's *fingerprint* to them. To see this fingerprint, use the `-kvc` command:

```
pgp -kvc userid [keyring]
```

This will display the key with the 16-byte digest of the public key components. Read this 16-byte fingerprint to the key's owner on the phone, while she checks it against her own, using the same -kvc command at her end.

You can both verify each other's keys this way, and then you can 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 E-mail is not the best way to verify the key, because E-mail can be intercepted and modified. It's 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 E-mail, and the key fingerprint via a voice telephone conversation. Some people distribute their key fingerprint on their business cards, which looks really cool.

For current versions of PGP, the key fingerprint is computed using the MD5 hash function. A future version of PGP will optionally use a new and different hash function, SHA, instead of MD5.

If you don't know me, please don't call me to verify my key over the phone—I get too many calls like that. Since every PGP user has a copy of my public key, no one could tamper with all the copies that are out there. The discrepancy would soon be noticed by someone who checked it from more than one source, and word would soon get out on the Internet.

For those of you who want to verify my public key (included in the standard PGP release package), here are the particulars:

```
UserID: "Philip R. Zimmermann <prz@acm.org>"
Key Size: 1024 bits;  Creation date: 21 May 1993;  KeyID: C7A966DD
Key fingerprint:  9E 94 45 13 39 83 5F 70  7B E7 D8 ED C4 BE 5A A6
```

The information printed above conceivably could still be tampered with in the electronic distribution of the PGP User's Guide. But if you read this in the printed version of the manual, available in bookstores from MIT Press, it's a safe bet that it really is my own key's fingerprint.

3.14 Handling large public keyrings

PGP was originally designed for handling small personal keyrings for keeping all your friends on, like a personal rolodex. A couple hundred keys is a reasonable size for such a keyring. But as PGP has become more popular, people are now trying to add other large keyrings to their own keyring. Sometimes this involves adding thousands of keys to your keyring. PGP, in its present form, cannot perform this operation in a reasonable period of time, while you wait at your keyboard. Not for huge keyrings.

You may want to add a huge "imported" keyring to your own keyring, because you are only interested in a few dozen keys on the bigger keyring you are bringing in. If that's all you want from the other keyring, it would be more efficient if you extract the few keys you need from the big foreign keyring, and then add just these few keys to your own keyring. Use the -kx command to extract them from the foreign keyring, specifying the keyring name on the command line. Then add these extracted keys to your own keyring.

The real solution is to improve PGP to use advanced database techniques to manage large keyrings efficiently. We are working on this, and should have it done Real Soon Now. Until this happens, you will just have to use smaller keyrings, or be patient.

3.15 Using PGP as a Unix-style filter

Unix fans are accustomed to using Unix *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*. PGP can operate in this mode. If you don't understand what this means, then you probably don't need this feature.

To use a Unix-style filter mode, reading from standard input and writing to standard output, add the -f option, like so:

```
pgp -feast her_userid <inputfile >outputfile
```

This feature makes it easier to make PGP work with electronic mail applications.

When using PGP in filter mode to decrypt a ciphertext file, you may find it useful to use the PGPPASS environmental variable to hold the pass phrase, so that you won't be prompted for it. The PGPPASS feature is explained below.

3.16 Suppressing unnecessary questions: BATCHMODE

With the BATCHMODE flag enabled on the command line, PGP will not ask any unnecessary questions or prompt for alternate filenames. Here is an example of how to set this flag:

```
pgp +batchmode cipherfile
```

This is useful for running PGP non-interactively from Unix shell scripts or MSDOS batch files. Some key management commands still need user interaction even when BATCHMODE is on, so shell scripts may need to avoid them.

BATCHMODE may also be enabled to check the validity of a signature on a file. If there was no signature on the file, the exit code is 1. If it had a signature that was good, the exit code is 0.

3.17 Force "Yes" answer to confirmation questions: FORCE

This command-line flag makes PGP assume `yes` for the user response to the confirmation request to overwrite an existing file, or when removing a key from the keyring via the `-kr` command. Here is an example of how to set this flag:

```
pgp +force cipherfile
```

or:

```
pgp -kr +force Smith
```

This feature is useful for running PGP non-interactively from a Unix shell script or MSDOS batch file.

3.18 PGP returns exit status to the shell

To facilitate running PGP in *batch* mode, such as from an MSDOS *.bat* file or from a Unix shell script, PGP returns an error exit status to the shell. An exit status code of zero means normal exit, while a nonzero exit status indicates some kind of error occurred. Different error exit conditions return different exit status codes to the shell.

3.19 Environmental variable for pass phrase

Normally, PGP prompts the user to type a pass phrase whenever PGP needs a pass phrase to unlock a secret key. But it is possible to store the pass phrase in an environmental variable from your operating system's command shell. The environmental variable PGPPASS can be used to hold the pass phrase that PGP will attempt to use first. If the pass phrase stored in PGPPASS is incorrect, PGP recovers by prompting the user for the correct pass phrase.

For example, on MSDOS, the shell command:

```
SET PGPPASS=zaphod beeblebrox for president
```

would eliminate the prompt for the pass phrase if the pass phrase were indeed *"zaphod beeblebrox for president"*.

This dangerous feature makes your life more convenient if you have to regularly deal with a large number of incoming messages addressed to your secret key, by eliminating the need for you to repeatedly type in your pass phrase every time you run PGP.

I added this feature because of popular demand. However, this is a somewhat dangerous feature, because it keeps your precious pass phrase stored somewhere other than just in your brain. Even worse, if you are particularly reckless, it may even be stored on a disk on the same computer as your secret key. It would be particularly dangerous and stupid if you were to install this command in a batch or script file, such as the MSDOS AUTOEXEC.BAT file. Someone could come along on your lunch hour and steal both your secret key ring and the file containing your pass phrase.

I can't emphasize the importance of this risk enough. If you are contemplating using this feature, be sure to read the sections "Exposure on Multi-user Systems" and "How to Protect Secret Keys from Disclosure" in this volume and in the Essential Topics volume of the PGP User's Guide.

If you must use this feature, the safest way to do it would be to just manually type in the shell command to set PGPPASS every time you boot your machine to start using PGP, and then erase it or turn off your machine when you are done. And you should definitely never do it in an environment where someone else may have access to your machine. Someone could come along and simply ask your computer to display the contents of PGPPASS.

Sometimes you want to pass the pass phrase into PGP from another application, such as an E-mail package. In some cases, it may not always be desirable to use the PGPPASS variable for that purpose. There is another way to pass your pass phrase into PGP from another application. Use the `-z` command line option. This option is designed primarily for invoking PGP from inside an E-mail package. The pass phrase follows the -z option on the command line. There are risks associated with using this approach, similar to those risks described above for using the PGPPASS variable.

3.20 Setting parameters in the PGP configuration file

PGP has a number of user-settable parameters that can be defined in a special PGP configuration text file called *config.txt*, in the directory pointed to by the shell environmental variable PGPPATH. Having a configuration file enables the user to define various flags and parameters for PGP without the burden of having to always define these parameters in the PGP command line.

The filename *config.txt* has been in use for a long time by PGP, but some folks have pointed out that it may be at odds with naming conventions for configuration files for specific operating systems. Accordingly, PGP now tries to open this filename only after first trying to open the file *.pgprc* on Unix platforms, and *pgp.ini* on other platforms, in the same directory that PGP would look for *config.txt*.

Configuration parameters may be assigned integer values, character string values, or on/off values, depending on what kind of configuration parameter it is. A sample configuration file is provided with PGP, so you can see some examples.

In the configuration file, blank lines are ignored, as is anything following the '#' comment character. Keywords are not case-sensitive.

Here is a short sample fragment of a typical configuration file:

```
# 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
```

If some configuration parameters are not defined in the configuration file, or if there is no configuration file, or if PGP can't find the configuration file, the values for the configuration parameters default to some reasonable value.

Note that it is also possible to set these same configuration parameters directly from the PGP command line, by preceding the parameter setting with a `+` character. For example, the following two PGP commands produce the same effect:

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

or:

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

The following is a summary of the various parameters than may be defined in the configuration file.

3.20.1 TMP: directory pathname for temporary files

Default setting: `TMP = ""`

The configuration parameter TMP specifies what directory to use for PGP's temporary scratch files. The best place to put them is on a RAM disk, if you have one. That speeds things up quite a bit, and increases security somewhat. If TMP is undefined, the temporary files go in the current directory. If the shell environmental variable TMP is defined, PGP instead uses that to specify where the temporary files should go.

3.20.2 LANGUAGE: foreign language selector

Default setting: `LANGUAGE = "en"`

PGP displays various prompts, warning messages, and advisories to the user on the screen. For example, messages such as *"File not found."*, or *"Please enter your pass phrase:"*. These messages are normally in English. But it is possible to get PGP to display its messages to the user in other languages, without having to modify the PGP executable program.

A number of people in various countries have translated all of PGP's display messages, warnings, and prompts into their native languages. These hundreds of translated message strings have been placed in a special text file called *language.txt*, distributed with the PGP release. The messages are stored in this file in English, Spanish, Dutch, German, French, Italian, Russian, Latvian, and Lithuanian. Other languages may be added later.

The configuration parameter LANGUAGE specifies what language to display these messages in. LANGUAGE may be set to `en` for English, `es` for Spanish, `de` for German, `nl` for Dutch, `fr` for French, `it` for Italian, `ru` for Russian, `lt3` for Lithuanian, `lv` for Latvian, `esp` for Esperanto. For example, if this line appeared in the configuration file:

```
LANGUAGE = "fr"
```

PGP would select French as the language for its display messages. The default setting is English.

When PGP needs to display a message to the user, it looks in the *language.txt* file for the equivalent message string in the selected foreign language and displays that translated message to the user. If PGP can't find the language string file, or if the selected language is not in the file, or if that one phrase is not translated into the selected language in the file, or if that phrase is missing entirely from the file, PGP displays the message in English.

To conserve disk space, most foreign translations are not included in the standard PGP release package, but are available separately.

3.20.3 MYNAME: default user ID for making signatures

Default setting: `MYNAME = ""`

The configuration parameter MYNAME specifies the default user ID to use to select the secret key for making signatures. If MYNAME is not defined, the most recent secret key you installed on your secret key ring will be used. The user may also override this setting by specifying a user ID on the PGP command line with the -u option.

3.20.4 TEXTMODE: assuming plaintext is a text file

Default setting: `TEXTMODE = off`

The configuration parameter TEXTMODE is equivalent to the -t command line option. If enabled, it causes PGP to assume the plaintext is a text file, not a binary file, and converts it to dfncanonical text before encrypting it. Canonical text has a carriage return and a linefeed at the end of each line of text.

This mode will be automatically turned off if PGP detects that the plaintext file contains what it thinks is non-text binary data. If you intend to use PGP primarily for E-mail purposes, you should turn TEXTMODE=ON.

For VAX/VMS systems, the current version of PGP defaults TEXTMODE=ON.

For further details, see the section "Sending ASCII Text Files Across Different Machine Environments".

3.20.5 CHARSET: specifies local character set for text files

Default setting: `CHARSET = NOCONV`

Because PGP must process messages in many non-English languages with non-ASCII character sets, you may have a need to tell PGP what local character set your machine uses. This determines what character conversions are performed when converting plaintext files to and from canonical text format. This is only a concern if you are in a non-English non-ASCII environment.

The configuration parameter CHARSET selects the local character set. The choices are NOCONV (no conversion), LATIN1 (ISO 8859-1 Latin Alphabet 1), KOI8 (used by most Russian Unix systems), ALT_CODES (used by Russian MSDOS systems), ASCII, and CP850 (used by most western European languages on standard MSDOS PCs).

LATIN1 is the internal representation used by PGP for canonical text, so if you select LATIN1, no conversion is done. Note also that PGP treats KOI8 as LATIN1, even though it is a completely different character set (Russian), because trying to convert KOI8 to either LATIN1 or CP850 would be futile anyway. This means that setting CHARSET to NOCONV, LATIN1, or KOI8 are all equivalent to PGP.

If you use MSDOS and expect to send or receive traffic in western European languages, set `CHARSET = "CP850"`. This will make PGP convert incoming canonical text messages from LATIN1 to CP850 after decryption. If you use the -t (textmode) option to convert to canonical text, PGP will convert your CP850 text to LATIN1 before encrypting it.

For further details, see the section "Sending ASCII Text Files Across Different Machine Environments".

3.20.6 ARMOR: enable ASCII armor output

Default setting: `ARMOR = off`

The configuration parameter ARMOR is equivalent to the -a command line option. If enabled, it causes PGP to emit ciphertext or keys in ASCII Radix-64 format suitable for transporting through E-mail channels. Output files are named with the *.asc* extension.

If you intend to use PGP primarily for E-mail purposes, you should turn ARMOR=ON.

For further details, see the section "Sending Ciphertext Through E-mail Channels: Radix-64 Format" in the Essential Topics volume.

3.20.7 ARMORLINES: size of ASCII armor multipart files

Default setting: `ARMORLINES = 720`

When PGP creates a very large *.asc* radix-64 file for sending ciphertext or keys through the E-mail, it breaks the file up into separate chunks small enough to send through Internet mail utilities. Normally, Internet mailers prohibit files larger than about 50000 bytes, which means that if we restrict the number of lines to about 720, we'll be well within the limit. The file chunks are named with suffixes *.as1*, *.as2*, *.as3*, ...

The configuration parameter ARMORLINES specifies the maximum number of lines to make each chunk in a multipart *.asc* file sequence. If you set it to zero, PGP will not break up the file into chunks.

Fidonet E-mail files usually have an upper limit of about 32K bytes, so 450 lines would be appropriate for Fidonet environments.

For further details, see the section "Sending Ciphertext Through E-mail Channels: Radix-64 Format" in the Essential Topics volume.

3.20.8 KEEPBINARY: keep binary ciphertext files after decrypting

Default setting: `KEEPBINARY = off`

When PGP reads a *.asc* file, it recognizes that the file is in radix-64 format and will convert it back to binary before processing as it normally does, producing as a by-product a *.pgp* ciphertext file in binary form. After further processing to decrypt the *.pgp* file, the final output file will be in normal plaintext form.

You may want to delete the binary *.pgp* intermediate file, or you may want PGP to delete it for you automatically. You can still rerun PGP on the original *.asc* file.

The configuration parameter KEEPBINARY enables or disables keeping the intermediate *.pgp* file during decryption.

For further details, see the section "Sending Ciphertext Through E-mail Channels: Radix-64 Format" in the Essential Topics volume.

3.20.9 COMPRESS: enable compression

Default setting: `COMPRESS = on`

The configuration parameter COMPRESS enables or disables data compression before encryption. It is used mainly for debugging PGP. Normally, PGP attempts to compress the plaintext before it encrypts it. Generally, you should leave this alone and let PGP attempt to compress the plaintext.

3.20.10 COMPLETES_NEEDED: number of completely trusted introducers needed

Default setting: `COMPLETES_NEEDED = 1`

The configuration parameter COMPLETES_NEEDED specifies the minimum number of completely trusted introducers required to fully certify a public key on your public key ring. This gives you a way of tuning PGP's skepticism.

For further details, see the section "How Does PGP Keep Track of Which Keys are Valid?" in the Essential Topics volume.

3.20.11 MARGINALS_NEEDED: number of marginally trusted introducers needed

Default setting: `MARGINALS_NEEDED = 2`

The configuration parameter MARGINALS_NEEDED specifies the minimum number of marginally trusted introducers required to fully certify a public key on your public key ring. This gives you a way of tuning PGP's skepticism.

For further details, see the section "How Does PGP Keep Track of Which Keys are Valid?" in the Essential Topics volume.

3.20.12 CERT_DEPTH: how deep may introducers be nested

Default setting: `CERT_DEPTH = 4`

The configuration parameter CERT_DEPTH specifies how many levels deep you may nest introducers to certify other introducers to certify public keys on your public key ring. For example, If CERT_DEPTH is set to 1, there may only be one layer of introducers below your own ultimately-trusted key. If that were the case, you would be required to directly certify the public keys of all trusted introducers on your key ring. If you set CERT_DEPTH to 0, you could have no introducers at all, and you would have to directly certify each and every key on your public key ring in order to use it. The minimum CERT_DEPTH is 0, the maximum is 8.

For further details, see the section "How Does PGP Keep Track of Which Keys are Valid?" in the Essential Topics volume.

3.20.13 BAKRING: filename for backup secret keyring

Default setting: `BAKRING = ""`

All of the key certification that PGP does on your public key ring ultimately depends on your own ultimately-trusted public key (or keys). To detect any tampering of your public key ring, PGP must check that your own key has not been tampered with. To do this, PGP must compare your public key against a backup copy of your secret key on some tamper-resistant media, such as a write-protected floppy disk. A secret key contains all the information that your public key has, plus some secret components. This means PGP can check your public key against a backup copy of your secret key.

The configuration parameter BAKRING specifies what pathname to use for PGP's trusted backup copy of your secret key ring. On MSDOS, you could set it to *a:\secring.pgp* to point it at a write-protected backup copy of your secret key ring on your floppy drive. This check is performed only when you execute the PGP -kc option to check your whole public key ring.

If BAKRING is not defined, PGP will not check your own key against any backup copy.

For further details, see the sections "How to Protect Public Keys from Tampering" and "How Does PGP Keep Track of Which Keys are Valid?" in the Essential Topics volume.

3.20.14 PUBRING: filename for your public keyring

Default setting: `PUBRING = "$PGPPATH/pubring.pgp"`

You may want to keep your public keyring in a directory separate from your PGP configuration file in the directory specified by your $PGPPATH environmental variable. You may specify the full path and filename for your public keyring by setting the PUBRING parameter. For example, on an MSDOS system, you might want to keep your public keyring on a floppy disk by:

```
PUBRING = "a:pubring.pgp"
```

This feature is especially handy for specifying an alternative keyring on the command line.

3.20.15 SECRING: filename for your secret keyring

Default setting: `SECRING = "$PGPPATH/secring.pgp"`

You may want to keep your secret keyring in a directory separate from your PGP configuration file in the directory specified by your $PGPPATH environmental variable. This comes in handy for putting your secret keyring in a directory or device that is more protected than your public keyring. You may specify the full path and filename for your secret keyring by setting the SECRING parameter. For example, on an MSDOS system, you might want to keep your secret keyring on a floppy disk by:

```
SECRING = "a:secring.pgp"
```

3.20.16 RANDSEED: filename for random number seed

Default setting: `RANDSEED = "$PGPPATH/randseed.bin"`

You may want to keep your random number seed file (for generation of session keys) in a directory separate from your PGP configuration file in the directory specified by your $PGPPATH environmental variable. This comes in handy for putting your random number seed file in a directory or device that is more protected than your public keyring. You may specify the full path and filename for your random seed file by setting the RANDSEED parameter. For example, on an MSDOS system, you might want to keep it on a floppy disk by:

```
RANDSEED = "a:randseed.bin"
```

3.20.17 PAGER: selects shell command to display plaintext output

Default setting: `PAGER = ""`

PGP lets you view the decrypted plaintext output on your screen (like the Unix-style *more* command), without writing it to a file, if you use the -m (more) option while decrypting. This displays the decrypted plaintext display on your screen one screenful at a time.

If you prefer to use a fancier page display utility, rather than PGP's built-in one, you can specify the name of a shell command that PGP will invoke to display your plaintext output file. The configuration parameter PAGER specifies the shell command to invoke to display the file. For example, on MSDOS systems, you might want to use the popular shareware program *list.com* to display your plaintext message. Assuming you have a copy of *list.com*, you may set PAGER accordingly:

```
PAGER = "list"
```

However, if the sender specified that this file is for your eyes only, and may not be written to disk, PGP always uses its own built-in display function.

For further details, see the section "Displaying Decrypted Plaintext on Your Screen".

3.20.18 SHOWPASS: echo pass phrase to user

Default setting: `SHOWPASS = off`

Normally, PGP does not let you see your pass phrase as you type it in. This makes it harder for someone to look over your shoulder while you type and learn your pass phrase. But some typing-impaired people have problems typing their pass phrase without seeing what they are typing, and they may be typing in the privacy of their own homes. So they asked if PGP can be configured to let them see what they type when they type in their pass phrase.

The configuration parameter SHOWPASS enables PGP to echo your typing during pass phrase entry.

3.20.19 TZFIX: timezone adjustment

Default setting: `TZFIX = 0`

PGP provides timestamps for keys and signature certificates in Greenwich Mean Time (GMT), or Coordinated Universal Time (UTC), which means the same thing for our purposes. When PGP asks the system for the time of day, the system is supposed to provide it in GMT.

But sometimes, because of improperly configured MSDOS systems, the system time is returned in US Pacific Standard Time time plus 8 hours. Sounds weird, doesn't it? Perhaps because of some sort of US west-coast jingoism, MSDOS presumes local time is US Pacific time, and pre-corrects Pacific time to GMT. This adversely affects the behavior of the internal MSDOS GMT time function that PGP calls. However, if your MSDOS environmental variable TZ is already properly defined for your timezone, this corrects the misconception MSDOS has that the whole world lives on the US west coast.

The configuration parameter TZFIX specifies the number of hours to add to the system time function to get GMT, for GMT timestamps on keys and signatures. If the MSDOS environmental variable TZ is defined properly, you can leave TZFIX=0. Unix systems usually shouldn't need to worry about setting TZFIX at all. But if you are using some other obscure operating system that doesn't know about GMT, you may have to use TZFIX to adjust the system time to GMT.

On MSDOS systems that do not have TZ defined in the environment, you should make TZFIX=0 for California, -1 for Colorado, -2 for Chicago, -3 for New York, -8 for London, -9 for Amsterdam. In the summer, TZFIX should be manually decremented from these values. What a mess.

It would be much cleaner to set your MSDOS environmental variable TZ in your AUTOEXEC.BAT file, and not use the TZFIX correction. Then MSDOS gives you good GMT timestamps, and will handle daylight savings time adjustments for you. Here are some sample lines to insert into AUTOEXEC.BAT, depending on your time zone:

```
For Los Angeles:  SET TZ=PST8PDT
For Denver:       SET TZ=MST7MDT
For Arizona:      SET TZ=MST7
   (Arizona never uses 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 Aukland:      SET TZ=NZT-13
```

3.20.20 CLEARSIG: enable signed messages to be encapsulated as clear text

Default setting: `CLEARSIG = on`

Normally, unencrypted PGP signed messages have a signature certificate prepended in binary form. Also, the signed message is compressed, rendering the message unreadable to casual human eyes, even though the message is not actually encrypted. To send this binary data through a 7-bit E-mail channel, radix-64 ASCII armor is applied (see the ARMOR parameter). Even if PGP didn't compress the message, the ASCII armor would still render the message unreadable to human eyes. The recipient must use PGP to strip the armor off and decompress it before reading the message.

If the original plaintext message is in text (not binary) form, there is a way to send a signed message through an E-mail channel in such a way that the signed message is not compressed and the ASCII armor is applied only to the binary signature certificate, but not to the plaintext message. The CLEARSIG flag provides this useful feature, making it possible to generate a signed message that can be read with human eyes, without the aid of PGP. Of course, you still need PGP to actually check the signature.

The CLEARSIG flag is preset to `on` beginning with PGP version 2.5. 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 your config file has CLEARSIG turned off, you can turn it back on again directly on the command line, like so:

```
pgp -sta +clearsig=on message.txt
```

This message representation is analogous to the MIC-CLEAR message type used in Internet Privacy Enhanced Mail (PEM). It is important to 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 some E-mail 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. But since PEM lives under a similar vulnerability, it seems worth having this feature despite the risks.

Beginning with PGP version 2.2, trailing blanks are ignored on each line in calculating the signature for text in CLEARSIG mode.

3.20.21 VERBOSE: quiet, normal, or verbose messages

Default setting: `VERBOSE = 1`

VERBOSE may be set to 0, 1, or 2, depending on how much detail you want to see from PGP diagnostic messages. The settings are:

0: Display messages only if there is a problem. Unix fans wanted this "quiet mode" setting.

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. Besides, PGP doesn't have any problems, right?

3.20.22 INTERACTIVE: ask for confirmation for key adds

Default Setting: `INTERACTIVE = off`

Enabling this mode will mean that if you add a key file containing multiple keys to your key ring, PGP will ask for confirmation for each key before adding it to your key ring.

3.20.23 NOMANUAL: let PGP generate keys without the manual

Default Setting: `NOMANUAL = off`

It is important that the freeware version of PGP not be distributed without the user documentation, which normally comes with it in the standard release package. This manual contains important information for using PGP, as well as important legal notices. But some people have distributed previous versions of PGP without the manual, causing a lot of problems for a lot of people who get it. To discourage the distribution of PGP without the required documentation, PGP has been changed to require the PGP User's Guide to be found somewhere on your computer (like in your PGP directory) before PGP will let you generate a key pair. However, some users like to use PGP on tiny palmtop computers with limited storage capacity, so they like to run PGP without the documentation present on their systems. To satisfy these users, PGP can be made to relax its requirement that the manual be present, by enabling the NOMANUAL flag on the command line during key generation, like so:

```
pgp -kg +nomanual
```

The NOMANUAL flag can only be set on the command line, not in the config file. Since you must read this manual to learn how to enable this simple override feature, I hope this will still be effective in discouraging the distribution of PGP without the manual.

Some people may object to PGP insisting on finding the manual somewhere in the neighborhood to generate a key. They bristle against this seemingly authoritarian attitude. Some people have even modified PGP to defeat this feature, and redistributed their hotwired version to others. That creates problems for me. Before I added this feature, there were maimed versions of the PGP distribution package floating around that lacked the manual. One of them was uploaded to Compuserve, and was distributed to countless users who called me on the phone to ask me why such a complicated program had no manual. It spread out to BBS systems around the country. And a freeware distributor got hold of the package from Compuserve and enshrined it on CD-ROM, distributing thousands of copies without the manual. What a mess.

3.21 A peek under the hood

Let's take a look at a few internal features of PGP.

3.21.1 Random numbers

PGP uses a cryptographically strong pseudorandom number generator for creating temporary conventional session keys. The seed file for this is called *randseed.bin*. It too can be kept in whatever directory is indicated by the PGPPATH environmental variable. If this random seed file does not exist, it is automatically created and seeded with truly random numbers derived from timing your keystroke latencies.

This generator reseeds the disk file each time it is used by mixing in new key material partially derived with the time of day and other truly random sources. It uses the conventional encryption algorithm as an engine for the random number generator. The seed file contains both random seed material and random key material to key the conventional encryption engine for the random generator.

This random seed file should be at least slightly protected from disclosure, to reduce the risk of an attacker deriving your next or previous session keys. The attacker would have a very hard time getting anything useful from capturing this random seed file, because the file is cryptographically laundered before and after each use. Nonetheless, it seems prudent to at least try to keep it from falling into the wrong hands.

If you feel uneasy about trusting any algorithmically derived random number source however strong, keep in mind that you already trust the strength of the same conventional cipher to protect your messages. If it's strong enough for that, then it should be strong enough to use as a source of random numbers for temporary session keys. Note that PGP still uses truly random numbers from physical sources (mainly keyboard timings) to generate long-term public/secret key pairs.

3.21.2 PGP's conventional encryption algorithm

As described earlier, PGP "bootstraps" into a conventional single-key encryption algorithm by using a public key algorithm to encipher the conventional session key and then switching to fast conventional cryptography. So let's talk about this conventional encryption algorithm. It isn't the DES.

The Federal Data Encryption Standard (DES) used to be a good algorithm for most commercial applications. But the Government never did trust the DES to protect its own classified data, because the DES key length is only 56 bits, short enough for a brute force attack. Also, the full 16-round DES has been attacked with some success by Biham and Shamir using differential cryptanalysis, and by Matsui using linear cryptanalysis.

The most devastating practical attack on the DES was described at the Crypto '93 conference, where Michael Wiener of Bell Northern Research presented a paper on how to crack the DES with a special machine. He has fully designed and tested a chip that guesses 50 million DES keys per second until it finds the right one. Although he has refrained from building the real chips so far, he can get these chips manufactured for $10.50 each, and can build 57000 of them into a special machine for $1 million that can try every DES key in 7 hours, averaging a solution in 3.5 hours. $1 million can be hidden in the budget of many companies. For $10 million, it takes 21 minutes to crack, and for $100 million, just two minutes. With any major government's budget for examining DES traffic, it can be cracked in seconds. This means that straight 56-bit DES is now effectively dead for purposes of serious data security applications.

A possible successor to DES may be a variation known as *triple DES*, which uses two DES keys to encrypt three times, achieving an effective key space of 112 bits. But this approach is three times slower than normal DES. A future version of PGP may support triple DES as an option.

PGP does not use the DES as its conventional single-key algorithm to encrypt messages. Instead, PGP uses a different conventional single-key block encryption algorithm, called IDEA(tm).

For the cryptographically curious, the IDEA cipher has a 64-bit block size for the plaintext and the ciphertext. It uses a key size of 128 bits. It is based on the design concept of "mixing operations from different algebraic groups". It runs much faster in software than the DES. Like the DES, it can be used in cipher feedback (CFB) and cipher block chaining (CBC) modes. PGP uses it in 64-bit CFB mode.

The IPES/IDEA block cipher was developed at ETH in Zurich by James L. Massey and Xuejia Lai, and published in 1990. This is not a "home-grown" algorithm. Its designers have a distinguished reputation in the cryptologic community. Early published papers on the algorithm called it IPES (Improved Proposed Encryption Standard), but they later changed the name to IDEA (International Data Encryption Algorithm). So far, IDEA has resisted attack much better than other ciphers such as FEAL, REDOC-II, LOKI, Snefru and Khafre. And recent evidence suggests that IDEA is more resistant than the DES to Biham & Shamir's highly successful differential cryptanalysis attack. Biham and Shamir have been examining the IDEA cipher for weaknesses, without success. Academic cryptanalyst groups in Belgium, England, and Germany are also attempting to attack it, as well as the military services from several European countries. As this new cipher continues to attract attack efforts from the most formidable quarters of the cryptanalytic world, confidence in IDEA is growing with the passage of time.

Every once in a while, I get a letter from someone who has just learned the awful truth that PGP does not use pure RSA to encrypt bulk data. They are concerned that the whole package is weakened if we use a hybrid public-key and conventional scheme just to speed things up. After all, a chain is only as strong as its weakest link. They demand an explanation for this apparent "compromise" in the strength of PGP. This may be because they have been caught up in the public's reverence and awe for the strength and mystique of RSA, mistakenly believing that RSA is intrinsically stronger than any conventional cipher. Well, it's not.

People who work in factoring research say that the workload to exhaust all the possible 128-bit keys in the IDEA cipher would roughly equal the factoring workload to crack a 3100-bit RSA key, which is quite a bit bigger than the 1024-bit RSA key size that most people use for high security applications. Given this range of key sizes, and assuming there are no hidden weaknesses in the conventional cipher, the weak link in this hybrid approach is in the public key algorithm, not the conventional cipher.

It is not ergonomically practical to use pure RSA with large keys to encrypt and decrypt long messages. A 1024-bit RSA key would decrypt messages about 4000 times slower than the IDEA cipher. Absolutely no one does it that way in the real world. Many people less experienced in cryptography do not realize that the attraction of public key cryptography is not because it is intrinsically stronger than a conventional cipher– its appeal is because it helps you manage keys more conveniently.

Not only is RSA too slow to use on bulk data, but it even has certain weaknesses that can be exploited in some special cases of particular kinds of messages that are fed to the RSA cipher, even for large keys. These special cases can be avoided by using the hybrid approach of using RSA to encrypt random session keys for a conventional cipher, like PGP does. So the bottom line is this: Using pure RSA on bulk data is the wrong approach, period. It's too slow, it's not stronger, and may even be weaker. If you find a software application that uses pure RSA on bulk data, it probably means the implementor does not understand these issues, which could imply he doesn't understand other important concepts of cryptography.

3.21.3 Data compression

PGP normally compresses the plaintext before encrypting it. It's too late to compress it after it has been encrypted; encrypted data is incompressible. Data compression saves modem transmission time and disk space and more importantly strengthens cryptographic security. Most cryptanalysis techniques exploit redundancies found in the plaintext to crack the cipher. Data compression reduces this redundancy in the plaintext, thereby greatly enhancing resistance to cryptanalysis. It takes extra time to compress the plaintext, but from a security point of view it seems worth it, at least in my cautious opinion.

Files that are too short to compress or just don't compress well are not compressed by PGP.

If you prefer, you can use PKZIP to compress the plaintext before encrypting it. PKZIP is a widely-available and effective MSDOS shareware compression utility from PKWare, Inc. Or you can use ZIP, a PKZIP-compatible freeware compression utility on Unix and other systems, available from Jean-Loup Gailly. There is some advantage in using PKZIP or ZIP in certain cases, because unlike PGP's built-in compression algorithm, PKZIP and ZIP have the nice feature of compressing multiple files into a single compressed file, which is reconstituted again into separate files when decompressed. PGP will not try to compress a plaintext file that has already been compressed. After decrypting, the recipient can decompress the plaintext with PKUNZIP. If the decrypted plaintext is a PKZIP compressed file, PGP automatically recognizes this and advises the recipient that the decrypted plaintext appears to be a PKZIP file.

For the technically curious readers, the current version of PGP uses the freeware ZIP compression routines written by Jean-loup Gailly, Mark Adler, and Richard B. Wales. This ZIP software uses functionally-equivalent compression algorithms as those used by PKWare's new PKZIP 2.0. This ZIP compression software was selected for PGP mainly because of its free portable C source code availability, and because it has a really good compression ratio, and because it's fast.

Peter Gutmann has also written a nice compression utility called HPACK, available for free from many Internet FTP sites. It encrypts the compressed archives, using PGP data formats and key rings. He wanted me to mention that here.

3.21.4 Message digests and digital signatures

To create a digital signature, PGP encrypts with your secret key. But PGP doesn't actually encrypt your entire message with your secret key– that would take too long. Instead, PGP encrypts a "message digest".

The message digest is a compact (128 bit) "distillate" of your message, similar in concept to a checksum. You can also think of it as a "fingerprint" of the message. The message digest "represents" your message, such that if the message were altered in any way, a different message digest would be computed from it. This makes it possible to detect any changes made to the message by a forger. A message digest is computed using a cryptographically strong one-way hash function of the message. It would be computationally infeasible for an attacker to devise a substitute message that would produce an identical message digest. In that respect, a message digest is much better than a checksum, because it is easy to devise a different message that would produce the same checksum. But like a checksum, you can't derive the original message from its message digest.

A message digest alone is not enough to authenticate a message. The message digest algorithm is publicly known, and does not require knowledge of any secret keys to calculate. If all we did was attach a message digest to a message, then a forger could alter a message and simply attach a new message digest calculated from the new altered message. To provide real authentication, the sender has to encrypt (sign) the message digest with his secret key.

A message digest is calculated from the message by the sender. The sender's secret key is used to encrypt the message digest and an electronic timestamp, forming a digital signature, or signature certificate. The sender sends the digital signature along with the message. The receiver receives the message and the digital signature, and recovers the original message digest from the digital signature by decrypting it with the sender's public key. The receiver computes a new message digest from the message, and checks to see if it matches the one recovered from the digital signature. If it matches, then that proves the message was not altered, and it came from the sender who owns the public key used to check the signature.

A potential forger would have to either produce an altered message that produces an identical message digest (which is infeasible), or he would have to create a new digital signature from a different message digest (also infeasible, without knowing the true sender's secret key).

Digital signatures prove who sent the message, and that the message was not altered either by error or design. It also provides non-repudiation, which means the sender cannot easily disavow his signature on the message.

Using message digests to form digital signatures has other advantages besides being faster than directly signing the entire actual message with the secret key. Using message digests allows signatures to be of a standard small fixed size, regardless of the size of the actual message. It also allows the software to check the message integrity automatically, in a manner similar to using checksums. And it allows signatures to be stored separately from messages, perhaps even in a public archive, without revealing sensitive information about the actual messages, because no one can derive any message content from a message digest.

The message digest algorithm used here is the MD5 Message Digest Algorithm, placed in the public domain by RSA Data Security, Inc. MD5's designer, Ronald Rivest, writes this about MD5:

"It is conjectured that the difficulty of coming up with two messages having the same message digest is on the order of 2^64 operations, and that the difficulty of coming up with any message having a given message digest is on the order of 2^128 operations. The MD5 algorithm has been carefully scrutinized for weaknesses. It is, however, a relatively new algorithm and further security analysis is of course justified, as is the case with any new proposal of this sort. The level of security provided by MD5 should be sufficient for implementing very high security hybrid digital signature schemes based on MD5 and the RSA public-key cryptosystem."

3.22 Compatibility with previous versions of PGP

PGP version 2.6 can read anything produced by versions 2.3 through 2.7. However, because of a negotiated agreement between MIT and RSA Data Security, PGP 2.6 was programmed to change its behavior slightly on 1 September 1994, triggered by a built-in software timer. On that date, version 2.6 started producing a new and slightly different data format for messages, signatures and keys. PGP 2.6 will still be able to read and process messages, signatures, and keys produced under the old format, but it will generate the new format. This change is intended to discourage people from continuing to use the older (2.3a and earlier) versions of PGP, which Public Key Partners contends infringes its RSA patent (see the section on Legal Issues). ViaCrypt PGP (see the section Where to Get a Commercial Version of PGP), versions 2.4 and 2.7, avoids questions of infringement through Viacrypt's license arrangement with Public Key Partners. PGP 2.5 and 2.6 avoid questions of infringement by using the RSAREF(TM) Cryptographic Toolkit, under license from RSA Data Security, Inc.

Outside the United States, the RSA patent is not in force, so PGP users there are free to use implementations of PGP that do not rely on RSAREF and its restrictions. See the notes on foreign versions in the Legal Issues section later in this manual. It seems likely that any versions of PGP prepared outside the US will accept the new format, whose detailed description is available from MIT. If everyone upgrades before September 1994, or soon thereafter, there will be little interoperability problems.

This format change beginning with 2.6 is similar to the process that naturally happens when new features are added, causing older versions of PGP to be unable to read stuff from the newer PGP, while the newer version can still read the old stuff. The only difference is that this is a "legal upgrade", instead of a technical one. It's a worthwhile change, if it can achieve peace in our time.

According to ViaCrypt, which sells a commercial version of PGP, ViaCrypt PGP will evolve to maintain interoperability with new freeware versions of PGP.

There is a another change that effects interoperability with earlier versions of PGP. Unfortunately, due to data format limitations imposed by RSAREF, PGP 2.5 and 2.6 cannot interpret any messages or signatures made with PGP version 2.2 or earlier. Since we had no choice but to use the new data formats, because of the need to switch to RSAREF, we can't do anything about this problem.

Beginning with version 2.4 (which was ViaCrypt's first version) through at least 2.6, PGP does not allow you to generate RSA keys bigger than 1024 bits. The upper limit was always intended to be 1024 bits – there had to be some kind of upper limit, for performance and interoperability reasons. But because of a bug in earlier versions of PGP, it was possible to generate keys larger than 1024 bits. These larger keys caused interoperability problems between different older versions of PGP that used different arithmetic algorithms with different native word sizes. On some platforms, PGP choked on the larger keys. In addition to these older key size problems, the 1024-bit limit is now enforced by RSAREF. A 1024-bit key is very likely to be well out of reach of attacks by major governments. In a future version, PGP will support bigger keys.

In general, there is compatibility from version 2.0 upwards through 2.4. Because new features are added, older versions may not always be able to handle some files created with newer versions. Because of massive changes to all the algorithms and data structures, PGP version 2.0 (and later) is not even slightly compatible with PGP version 1.0, which no one uses anymore anyway.

Future versions of PGP may have to change the data formats for messages, signatures, keys and key rings, in order to provide important new features. We will endeavor to make future versions handle keys, signatures, and messages from this version, but this is not guaranteed. Future releases may provide conversion utilities to convert old keys, but you may have to dispose of old messages created with the old PGP. Also, this current version may not be able to read stuff produced from all future versions.

4 Vulnerabilities

No data security system is impenetrable. PGP can be circumvented in a variety of ways. In any data security system, you have to ask yourself if the information you are trying to protect is more valuable to your attacker than the cost of the attack. This should lead you to protecting yourself from the cheapest attacks, while not worrying about the more expensive attacks.

Some of the discussion that follows may seem unduly paranoid, but such an attitude is appropriate for a reasonable discussion of vulnerability issues.

4.1 Compromised pass phrase and secret key

Probably the simplest attack is if you leave your pass phrase for your secret key written down somewhere. If someone gets it and also gets your secret key file, they can read your messages and make signatures in your name.

Don't use obvious passwords that can be easily guessed, such as the names of your kids or spouse. If you make your pass phrase a single word, it can be easily guessed by having a computer try all the words in the dictionary until it finds your password. That's why a pass phrase is so much better than a password. A more sophisticated attacker may have his computer scan a book of famous quotations to find your pass phrase. An easy to remember but hard to guess pass phrase can be easily constructed by some creatively nonsensical sayings or very obscure literary quotes.

For further details, see the section "How to Protect Secret Keys from Disclosure" in the Essential Topics volume of the PGP User's Guide.

4.2 Public key tampering

A major vulnerability exists if public keys are tampered with. This may be the most crucially important vulnerability of a public key cryptosystem, in part because most novices don't immediately recognize it. The importance of this vulnerability, and appropriate hygienic countermeasures, are detailed in the section xlink"How to Protect Public Keys from Tampering"../pgpdoc1/how-to-protext-pk-from-tamp.html in the Essential Topics volume.

To summarize: When you use someone's public key, make certain it has not been tampered with. A new public key from someone else should be trusted only if you got it directly from its owner, or if it has been signed by someone you trust. Make sure no one else can tamper with your own public key ring. Maintain physical control of both your public key ring and your secret key ring, preferably on your own personal computer rather than on a remote timesharing system. Keep a backup copy of both key rings.

4.3 "Not quite deleted" files

Another potential security problem is caused by how most operating systems delete files. When you encrypt a file and then delete the original plaintext file, the operating system doesn't actually physically erase the data. It merely marks those disk blocks as deleted, allowing the space to be reused later. It's sort of like discarding sensitive paper documents in the paper recycling bin instead of the paper shredder. The disk blocks still contain the original sensitive data you wanted to erase, and will probably eventually be overwritten by new data at some point in the future. If an attacker reads these deleted disk blocks soon after they have been deallocated, he could recover your plaintext.

In fact this could even happen accidentally, if for some reason something went wrong with the disk and some files were accidentally deleted or corrupted. A disk recovery program may be run to recover the damaged files, but this often means some previously deleted files are resurrected along with everything else. Your confidential files that you thought were gone forever could then reappear and be inspected by whomever is attempting to recover your damaged disk. Even while you are creating the original message with a word processor or text editor, the editor may be creating multiple temporary copies of your text on the disk, just because of its internal workings. These temporary copies of your text are deleted by the word processor when it's done, but these sensitive fragments are still on your disk somewhere.

Let me tell you a true horror story. I had a friend, married with young children, who once had a brief and not very serious affair. She wrote a letter to her lover on her word processor, and deleted the letter after she sent it. Later, after the affair was over, the floppy disk got damaged somehow and she had to recover it because it contained other important documents. She asked her husband to salvage the disk, which seemed perfectly safe because she knew she had deleted the incriminating letter. Her husband ran a commercial disk recovery software package to salvage the files. It recovered the files alright, including the deleted letter. He read it, which set off a tragic chain of events.

The only way to prevent the plaintext from reappearing is to somehow cause the deleted plaintext files to be overwritten. Unless you know for sure that all the deleted disk blocks will soon be reused, you must take positive steps to overwrite the plaintext file, and also any fragments of it on the disk left by your word processor. You can overwrite the original plaintext file after encryption by using the PGP -w (wipe) option. You can take care of any fragments of the plaintext left on the disk by using any of the disk utilities available that can overwrite all of the unused blocks on a disk. For example, the Norton Utilities for MSDOS can do this.

Even if you overwrite the plaintext data on the disk, it may still be possible for a resourceful and determined attacker to recover the data. Faint magnetic traces of the original data remain on the disk after it has been overwritten. Special sophisticated disk recovery hardware can sometimes be used to recover the data.

4.4 Viruses and trojan horses

Another attack could involve a specially-tailored hostile computer virus or worm that might infect PGP or your operating system. This hypothetical virus could be designed to capture your pass phrase or secret key or deciphered messages, and covertly write the captured information to a file or send it through a network to the virus's owner. Or it might alter PGP's behavior so that signatures are not properly checked. This attack is cheaper than cryptanalytic attacks.

Defending against this falls under the category of defending against viral infection generally. There are some moderately capable anti-viral products commercially available, and there are hygienic procedures to follow that can greatly reduce the chances of viral infection. A complete treatment of anti-viral and anti-worm countermeasures is beyond the scope of this document. PGP has no defenses against viruses, and assumes your own personal computer is a trustworthy execution environment. If such a virus or worm actually appeared, hopefully word would soon get around warning everyone.

Another similar attack involves someone creating a clever imitation of PGP that behaves like PGP in most respects, but doesn't work the way it's supposed to. For example, it might be deliberately crippled to not check signatures properly, allowing bogus key certificates to be accepted. This "Trojan horse" version of PGP is not hard for an attacker to create, because PGP source code is widely available, so anyone could modify the source code and produce a lobotomized zombie imitation PGP that looks real but does the bidding of its diabolical master. This Trojan horse version of PGP could then be widely circulated, claiming to be from me. How insidious.

You should make an effort to get your copy of PGP from a reliable source, whatever that means. Or perhaps from more than one independent source, and compare them with a file comparison utility.

There are other ways to check PGP for tampering, using digital signatures. If someone you trust signs the executable version of PGP, vouching for the fact that it has not been infected or tampered with, you can be reasonably sure that you have a good copy. You could use an earlier trusted version of PGP to check the signature on a later suspect version of PGP. But this will not help at all if your operating system is infected, nor will it detect if your original copy of PGP.EXE has been maliciously altered in such a way as to compromise its own ability to check signatures. This test also assumes that you have a good trusted copy of the public key that you use to check the signature on the PGP executable.

I recommend you not trust your copy of PGP unless it was originally distributed by MIT or ViaCrypt, or unless it comes with a digitally signed endorsement from me. Every new version comes with one or more digital signatures in the distribution package, signed by the originator of that release package. This is usually someone representing MIT or ViaCrypt, or whoever released that version. Check the signatures on the version that you get. I have actually seen several bogus versions of PGP distribution packages, even from apparantly reliable freeware distribution channels such as CD-ROM distributors and Compuserve. Always check the signature when you get a new version.

4.5 Physical security breach

A physical security breach may allow someone to physically acquire your plaintext files or printed messages. A determined opponent might accomplish this through burglary, trash-picking, unreasonable search and seizure, or bribery, blackmail or infiltration of your staff. Some of these attacks may be especially feasible against grassroots political organizations that depend on a largely volunteer staff. It has been widely reported in the press that the FBI's COINTELPRO program used burglary, infiltration, and illegal bugging against antiwar and civil rights groups. And look what happened at the Watergate Hotel.

Don't be lulled into a false sense of security just because you have a cryptographic tool. Cryptographic techniques protect data only while it's encrypted– direct physical security violations can still compromise plaintext data or written or spoken information.

This kind of attack is cheaper than cryptanalytic attacks on PGP.

4.6 Tempest attacks

Another kind of attack that has been used by well-equipped opponents involves the remote detection of the electromagnetic signals from your computer. This expensive and somewhat labor-intensive attack is probably still cheaper than direct cryptanalytic attacks. An appropriately instrumented van can park near your office and remotely pick up all of your keystrokes and messages displayed on your computer video screen. This would compromise all of your passwords, messages, etc. This attack can be thwarted by properly shielding all of your computer equipment and network cabling so that it does not emit these signals. This shielding technology is known as "Tempest", and is used by some Government agencies and defense contractors. There are hardware vendors who supply Tempest shielding commercially, although it may be subject to some kind of Government licensing. Now why do you suppose the Government would restrict access to Tempest shielding?

4.7 Exposure on multi-user systems

PGP was originally designed for a single-user MSDOS machine under your direct physical control. I run PGP at home on my own PC, and unless someone breaks into my house or monitors my electromagnetic emissions, they probably can't see my plaintext files or secret keys.

But now PGP also runs on multi-user systems such as Unix and VAX/VMS. On multi-user systems, there are much greater risks of your plaintext or keys or passwords being exposed. The Unix system administrator or a clever intruder can read your plaintext files, or perhaps even use special software to covertly monitor your keystrokes or read what's on your screen. On a Unix system, any other user can read your environment information remotely by simply using the Unix *ps* command. Similar problems exist for MSDOS machines connected on a local area network. The actual security risk is dependent on your particular situation. Some multi-user systems may be safe because all the users are trusted, or because they have system security measures that are safe enough to withstand the attacks available to the intruders, or because there just aren't any sufficiently interested intruders. Some Unix systems are safe because they are only used by one user– there are even some notebook computers running Unix. It would be unreasonable to simply exclude PGP from running on all Unix systems.

PGP is not designed to protect your data while it is in plaintext form on a compromised system. Nor can it prevent an intruder from using sophisticated measures to read your secret key while it is being used. You will just have to recognize these risks on multi-user systems, and adjust your expectations and behavior accordingly. Perhaps your situation is such that you should consider running PGP only on an isolated single-user system under your direct physical control. That's what I do, and that's what I recommend.

4.8 Traffic analysis

Even if the attacker cannot read the contents of your encrypted messages, he may be able to infer at least some useful information by observing where the messages come from and where they are going, the size of the messages, and the time of day the messages are sent. This is analogous to the attacker looking at your long distance phone bill to see who you called and when and for how long, even though the actual content of your calls is unknown to the attacker. This is called traffic analysis. PGP alone does not protect against traffic analysis. Solving this problem would require specialized communication protocols designed to reduce exposure to traffic analysis in your communication environment, possibly with some cryptographic assistance.

4.9 Protecting against bogus timestamps

A somewhat obscure vulnerability of PGP involves dishonest users creating bogus timestamps on their own public key certificates and signatures. You can skip over this section if you are a casual user and aren't deeply into obscure public key protocols.

There's nothing to stop a dishonest user from altering the date and time setting of his own system's clock, and generating his own public key certificates and signatures that appear to have been created at a different time. He can make it appear that he signed something earlier or later than he actually did, or that his public/secret key pair was created earlier or later. This may have some legal or financial benefit to him, for example by creating some kind of loophole that might allow him to repudiate a signature.

I think this problem of falsified timestamps in digital signatures is no worse than it is already in handwritten signatures. Anyone may write a date next to their handwritten signature on a contract with any date they choose, yet no one seems to be alarmed over this state of affairs. In some cases, an "incorrect" date on a handwritten signature might not be associated with actual fraud. The timestamp might be when the signator asserts that he signed a document, or maybe when he wants the signature to go into effect.

In situations where it is critical that a signature be trusted to have the actual correct date, people can simply use notaries to witness and date a handwritten signature. The analog to this in digital signatures is to get a trusted third party to sign a signature certificate, applying a trusted timestamp. No exotic or overly formal protocols are needed for this. Witnessed signatures have long been recognized as a legitimate way of determining when a document was signed.

A trustworthy Certifying Authority or notary could create notarized signatures with a trustworthy timestamp. This would not necessarily require a centralized authority. Perhaps any trusted introducer or disinterested party could serve this function, the same way real notary publics do now. When a notary signs other people's signatures, it creates a signature certificate of a signature certificate. This would serve as a witness to the signature the same way real notaries now witness handwritten signatures. The notary could enter the detached signature certificate (without the actual whole document that was signed) into a special log controlled by the notary. Anyone can read this log. The notary's signature would have a trusted timestamp, which might have greater credibility or more legal significance than the timestamp in the original signature.

There is a good treatment of this topic in Denning's 1983 article in IEEE Computer (see references). Future enhancements to PGP might have features to easily manage notarized signatures of signatures, with trusted timestamps.

4.10 Cryptanalysis

An expensive and formidable cryptanalytic attack could possibly be mounted by someone with vast supercomputer resources, such as a Government intelligence agency. They might crack your RSA key by using some new secret factoring breakthrough. Perhaps so, but it is noteworthy that the US Government trusts the RSA algorithm enough in some cases to use it to protect its own nuclear weapons, according to Ron Rivest. And civilian academia has been intensively attacking it without success since 1978.

Perhaps the Government has some classified methods of cracking the IDEA(tm) conventional encryption algorithm used in PGP. This is every cryptographer's worst nightmare. There can be no absolute security guarantees in practical cryptographic implementations.

Still, some optimism seems justified. The IDEA algorithm's designers are among the best cryptographers in Europe. It has had extensive security analysis and peer review from some of the best cryptanalysts in the unclassified world. It appears to have some design advantages over the DES in withstanding differential and linear cryptanalysis, which have both been used to crack the DES.

Besides, even if this algorithm has some subtle unknown weaknesses, PGP compresses the plaintext before encryption, which should greatly reduce those weaknesses. The computational workload to crack it is likely to be much more expensive than the value of the message.

If your situation justifies worrying about very formidable attacks of this caliber, then perhaps you should contact a data security consultant for some customized data security approaches tailored to your special needs. Boulder Software Engineering, whose address and phone are given at the end of this document, can provide such services.

In summary, without good cryptographic protection of your data communications, it may have been practically effortless and perhaps even routine for an opponent to intercept your messages, especially those sent through a modem or E-mail system. If you use PGP and follow reasonable precautions, the attacker will have to expend far more effort and expense to violate your privacy.

If you protect yourself against the simplest attacks, and you feel confident that your privacy is not going to be violated by a determined and highly resourceful attacker, then you'll probably be safe using PGP. PGP gives you Pretty Good Privacy.

5 Legal issues

5.1 Trademarks, copyrights, and warranties

"PGP", "Pretty Good Privacy", "Phil's Pretty Good Software", and the "Pretty Good" label for computer software and hardware products are all trademarks of Philip R. Zimmermann.

PGP is © Copyright Philip R. Zimmermann, 1990-1994. All rights reserved. The PGP User's Guide is also copyright Philip Zimmermann, 1990-1994. All rights reserved. These rights include but are not limited to any foreign language translations of the manual or the software, and all derivative works of both.

MIT may have a copyright on the particular software distribution package that they distribute from the MIT FTP site. This copyright on the "compilation" of the distribution package in no way implies that MIT has a copyright on PGP itself, or its user documentation.

The author assumes no liability for damages resulting from the use of this software, even if the damage results from defects in this software, and makes no representations concerning the merchantability of this software or its suitability for any specific purpose. It is provided "as is" without express or implied warranty of any kind. Because certain actions may delete files or render them unrecoverable, the author assumes no responsibility for the loss or modification of any data.

5.2 Patent rights on the algorithms

The RSA public key cryptosystem was developed at MIT, which holds a patent on it (U.S. patent #4,405,829, issued 20 Sep 1983). A company in California called Public Key Partners (PKP) holds the exclusive commercial license to sell and sub-license the RSA public key cryptosystem. MIT distributes a freeware version of PGP under the terms of the RSAREF license from RSA Data Security, Inc. (RSADSI).

At the time of this writing (September 1994), it appears that PKP may be breaking up soon, in which case the patents they hold may fall into other hands. The RSA patent may end up with RSADSI.

Non-US users of earlier versions of PGP should note that the RSA patent does not apply outside the US, and at least at the time of this writing, the author is not aware of any RSA patent in any other country. Federal agencies may use the RSA algorithm, because the Government paid for the development of RSA with grants from the National Science Foundation and the Navy. But despite the fact of Government users having free access to the RSA algorithm, Government use of PGP has additional restrictions imposed by the agreement I have with ViaCrypt, as explained later.

I wrote my PGP software from scratch, with my own independently developed implementation of the RSA algorithm. Before publishing PGP in 1991, I got a formal written legal opinion from a patent attorney with extensive experience in software patents. I'm convinced that publishing PGP the way I did does not violate patent law.

Not only did PKP acquire the exclusive patent rights for the RSA cryptosystem, but they also acquired the exclusive rights to three other patents covering other public key schemes invented by others at Stanford University, also developed with federal funding. This one company claims to have a legal lock in the USA on nearly all practical public key cryptosystems. They even appear to be claiming patent rights on the very concept of public key cryptography, regardless of what clever new original algorithms are independently invented by others. I find such a comprehensive monopoly troubling, because I think public key cryptography is destined to become a crucial technology in the protection of our civil liberties and privacy in our increasingly connected society. At the very least, it places these vital tools at risk by affording to the Government a single pressure point of influence.

Beginning with PGP version 2.5 (distributed by MIT, the holders of the original RSA patent), the freeware version of PGP uses the RSAREF subroutine library to perform its RSA calculations, under the RSAREF license, which allows noncommercial use in the USA. RSAREF is a subroutine package from RSA Data Security Inc, that implements the RSA algorithm. The RSAREF subroutines are used instead of PGP's original subroutines to implement the RSA functions in PGP. See the RSAREF license for terms and conditions of use of RSAREF applications.

PGP 2.5 was released by MIT for a brief test period in May, 1994 before releasing 2.6. PGP 2.5 was released under the 16 March, 1994 RSAREF license, which is a perpetual license, so it may legally be used forever in the US. But it would be better for PGP's legal and political future for users in the United States to upgrade to version 2.6 or later to facilitate the demise of PGP 2.3a and earlier versions. Also, PGP 2.5 has bugs that are corrected in 2.6, and 2.5 will not read the new data format after September 1, 1994. (See the section on Compatibility with Previous and Future Versions of PGP.)

The PGP 2.0 release was a joint effort of an international team of software engineers, implementing enhancements to the original PGP with design guidance from me. It was released by Branko Lankester in The Netherlands and Peter Gutmann in New Zealand, out of reach of US patent law. Although released only in Europe and New Zealand, it spontaneously spread to the USA without help from me or the PGP development team.

The IDEA(tm) conventional block cipher used by PGP is covered by a patent in Europe, held by ETH and a Swiss company called Ascom-Tech AG. The US Patent number is 5,214,703, and the European patent number is EP 0 482 154 B1. IDEA(tm) is a trademark of Ascom-Tech AG. There is no license fee required for noncommercial use of IDEA. Commercial users of IDEA may obtain licensing details from Dieter Profos, Ascom Tech AG, Teleservices Section, Postfach 151, 4502 Solothurn, Switzerland, Tel +41 65 242885, Fax +41 65 235761.

Ascom-Tech AG has granted permission for the freeware version PGP to use the IDEA cipher in non-commercial uses, everywhere. In the US and Canada, all commercial or Government users must obtain a licensed version from ViaCrypt, who has a license from Ascom-Tech for the IDEA cipher.

Ascom-Tech has recently been changing its policies regarding the use of IDEA in PGP for commercial use outside the US, and that policy still seems to be in flux. They tell me that their current thinking is as follows: They will allow commercial users of PGP outside the US or Canada to use IDEA in PGP without paying royalties to Ascom-Tech, because it is not currently possible for commercial users to buy a licensed version of PGP outside the US or Canada. If the legal situation in the USA changes in the future, so that users outside the US or Canada can buy a licensed version of PGP (either from ViaCrypt, or from me, or from a foreign enterprise licensed by me), then Ascom-Tech will begin enforcing its patent licensing policies on commercial users who are in a position to buy a licensed version of PGP. To get a more up-to-date report on this, contact Ascom-Tech AG.

The ZIP compression routines in PGP come from freeware source code, with the author's permission. I'm not aware of any patents on the compression algorithms used in the ZIP routines.

5.3 Freeware status and restrictions

PGP is not shareware, it's freeware. Published as a community service. Giving PGP away for free will encourage far more people to use it, which will have a greater social impact. Feel free to disseminate the complete unmodified PGP release package as widely as possible, but be careful not to violate U.S. export controls if you live in the USA. Give it to all your friends. If you have access to any electronic Bulletin Board Systems, please upload the complete PGP executable object release package to as many BBS's as possible.

You may also disseminate the source code release package. PGP's source code is published to assist public scrutiny of PGP to show that it has no hidden weaknesses or back doors, and to help people to find bugs and report them. Recompile it and port it to new target machines. Experiment with the code and learn from it.

I place no restraints on your modifying the source code for your own use. However, do not distribute a modified version of PGP under the name "PGP" without first getting permission from me. Please respect this restriction. PGP's reputation for cryptographic integrity depends on maintaining strict quality control on PGP's cryptographic algorithms and protocols. Beyond that, ad hoc "improvements" to PGP can affect interoperability, which creates user confusion and compatability problems that could damage PGP's (and my own) reputation and undermine the good will earned by the PGP trademark.

This has already started to happen, which is why I'm making a point of it here. This creates technical support headaches, and I get phone calls from confused users who run into problems either because they have a mutant strain of PGP, or are trying to process a key, signature, or message that came from an incompatible mutant strain of PGP. The source code to PGP was not published to help spawn these mutant strains.

If you want to distribute a modified version of PGP, or use a modified version to send messages to other people, you should name the program in such a way that no one could mistake it for PGP. The messages, signatures, and keys it produces must also be labeled in such a way that no one could mistake them for material produced by PGP. If you feel you must modify your copy of PGP, and there is any chance that the modified version could escape into the environment, please contact me first to discuss some easy methods for how to prevent people from confusing your version with the standard PGP. Perhaps we'll even decide that your changes are appropriate for incorporating into the standard PGP release.

Also, you should note that official executable versions of PGP are always released signed by the PGP developers, so you can verify their authenticity. If you find a corrupted copy of PGP, or notice one being distributed, please contact the people doing the distribution and suggest that they replace this with an authentic version.

Some older versions of PGP were published under the terms of the General Public License (GPL), a license designed by the Free Software Foundation to protect the status of free software. Newer freeware versions of PGP are no longer published under the GPL. The RSAREF licensing terms are more stringent than those of the GPL. But even if a version of PGP is published without RSAREF, in a situation or place where the RSA patent does not apply, I still do not want the GPL to apply to PGP, for a variety of reasons, not the least of which is because the GPL is not optimal for protecting PGP from being republished with ad-hoc "improvements".

Outside the United States, the RSA patent is not in force, so PGP users there are free to use implementations of PGP that do not rely on RSAREF and its restrictions. Canadians may use PGP without using RSAREF, and there are legal ways to export PGP to Canada. In Canada, where RSAREF is not needed, it is easy to modify and recompile the current PGP source code to perform the RSA calculations without using the RSAREF library, just as it was done in PGP 2.3a. In such a case, this modified PGP may be re-released under the identical licensing terms as the current official freeware PGP release, but without the RSAREF-specific restrictions. It may not be re-released under the GPL, as certain older versions were. And this manual must accompany it. That modified version of PGP may not be used in environments where RSAREF would be needed.

5.4 Restrictions on Commercial Use of PGP

The freeware version of PGP is for personal, non-commercial use. For commercial use in the USA or Canada, contact ViaCrypt in Phoenix, Arizona (phone 602 944-0773, or email viacrypt@acm.org).

I made an agreement with ViaCrypt in the summer of 1993 to license the exclusive commercial rights to PGP, so that there would be a way for corporations to use PGP without risk of a patent infringement lawsuit from PKP. For PGP to succeed in the long term as a viable industry standard, the legal stigma associated with the RSA patent rights had to be resolved. ViaCrypt had already obtained a patent license from PKP to make, use, and sell products that practice the RSA patents. ViaCrypt offered a way out of the patent quagmire for PGP to penetrate the corporate environment. They could sell a fully-licensed version of PGP, but only if I licensed it to them under these terms. So we entered into an agreement to do that, opening the door for PGP's future in the commercial sector, which was necessary for PGP's long-term political future.

Therefore, regardless of the complexities and partially overlapping restrictions from all the other terms and conditions imposed by the various patent and copyright licenses (RSA, RSAREF, and IDEA) from various third parties, an additional overriding restriction on PGP usage is imposed by my own agreement with ViaCrypt: The freeware version of PGP is only for personal, non-commercial use – all other users in the USA and Canada must obtain a fully licensed version of PGP from ViaCrypt. The restrictions imposed by my agreement with ViaCrypt do not apply outside the USA or Canada.

Finally, if you want to turn PGP into a commercial product and make money selling it, then we must agree on a way for me to also make money on it. If you use PGP in such a manner that you must pay patent royalties or any other software licensing fees to the patent holders for any cryptographic algorithms used by PGP, then we must agree on a way for me to also be paid in some manner. Buying PGP from ViaCrypt is one way to meet this requirement.

5.5 Other Licensing Restrictions

Under no circumstances may PGP be distributed without the PGP documentation, including this PGP User's Guide. And, assuming this is an RSAREF version of PGP, the RSAREF license agreement must be kept with it. You must also keep the copyright, patent, and trademark notices on PGP and its documentation.

The standard freeware PGP release is primarily distributed in electronic form, as a single compressed archive file, containing a collection of files in a "shrink-wrapped" package. This package should not be broken up and the components separately distributed – in the interests of quality control, we want to make it difficult for users to obtain PGP without getting the full release package.

5.6 Distribution

In the USA, PGP is available for free from the Massachusetts Institute of Technology, under the restrictions described above.

The primary release site for PGP is the Massachusetts Institute of Technology, at their FTP site *net-dist.mit.edu*, in the /pub/PGP directory. You may obtain free copies or updates to PGP from this site, or any other Internet FTP site or BBS that PGP has spread to. Don't ask me for a copy directly from me, especially if you live outside the US or Canada. I recommend that you not use any modified version of PGP that comes from any other source, other than MIT, ViaCrypt, or me, unless it is accompanied by a signed endorsement from me personally. You can get the official release software from many other distribution sites "downstream" from MIT. Hopefully, all these other sites are adhering to US export controls.

The PGP version 2.6.2 executable object release package for MSDOS contains the PGP executable software, documentation, RSAREF license, sample key rings including my own public key, and signatures for the software and this manual, all in one PKZIP compressed file called pgp262.zip. The PGP source release package for MSDOS contains all the C source files in one PKZIP compressed file called pgp262s.zip. The filename for the release package is derived from the version number of the release.

5.7 Export Controls

The U.S. Government has made it illegal in most cases to export good cryptographic technology, and that may include PGP. They regard this kind of software just like they regard munitions. This is determined not by legislation, but by administrative policies of the State Department, Defense Department and Commerce Department.

The U.S. Government is using export restrictions as a means of suppressing both domestic and foreign availability of cryptographic technology. In particular, it is trying to suppress the emergence of an international standard for cryptographic protocols, until it can establish the Escrowed Encryption Standard (the Clipper chip) as the dominant standard.

Any export restrictions on PGP are imposed by the US Government. This does not imply that I or MIT agree with these restrictions. We just comply with them. We do not impose additional licensing restrictions of our own on the use of PGP outside of the US, other than those restrictions that already apply inside the US. PGP may be subject to export controls. Anyone wishing to export it should first consult the State Department's Office of Defense Trade Controls.

I will not export this software out of the US or Canada in cases when it is illegal to do so under US controls, and I urge other people not to export it on their own. If you live outside the US or Canada, I urge you not to violate US export laws by getting any version of PGP in a way that violates those laws. Since thousands of domestic users got the first version after its initial publication, it somehow leaked out of the US and spread itself widely abroad, like dandelion seeds blowing in the wind.

Starting with PGP version 2.0 through version 2.3a, the release point of the software has been outside the US, on publicly-accessible computers in Europe. Each release was electronically sent back into the US and posted on publicly-accessible computers in the US by PGP privacy activists in foreign countries. There are some restrictions in the US regarding the import of munitions, but I'm not aware of any cases where this was ever enforced for importing cryptographic software into the US. I imagine that a legal action of that type would be quite a spectacle of controversy.

ViaCrypt PGP is sold in the United States and Canada and is not for export. The following language was supplied by the US Government to ViaCrypt for inclusion in the ViaCrypt PGP documentation: "PGP" is export restricted by the Office of Export Administration, United States Department of Commerce and the Offices of Defense Trade Controls and Munitions Control, United States Department of State. PGP cannot be exported or reexported, directly or indirectly, (a) without all export or reexport licenses and governmental approvals required by any applicable laws, or (b) in violation of any prohibition against the export or reexport of any part of PGP. The Government may take the position that the freeware PGP versions are also subject to those controls.

The freeware PGP versions 2.5 and 2.6 were released through a posting on a controlled FTP site maintained by MIT. This site has restrictions and limitations which have been used on other FTP sites to comply with export control requirements with respect to other encryption software such as Kerberos and software from RSA Data Security, Inc. I urge you not to do anything which would weaken those controls or facilitate any improper export of PGP.

Although PGP has become a worldwide de facto standard for E-mail encryption, and is widely available overseas, I still get calls from people outside the US who ask me if it is legal to use it in their own country, for versions that are already available there. Please don't contact me to ask me if it is legal to use PGP in your country if you live outside the US. That question is not up to me. I've got enough legal problems of my own with export control issues, without getting involved in giving you legal advice over my phone. It might even put me at some legal risk to simply answer a question like that for a foreigner. If this question concerns you, ask someone else, like a lawyer.

You may have a need to use PGP in a commercial application outside the US or Canada. Unfortunately, at the time of this writing, there is no current commercial source for PGP outside the US or Canada. I am trying to find a US-legal way to make a commercially licensed version available abroad, but right now the US export restrictions make that difficult without putting me at legal risk. This situation may change.

Some foreign governments impose serious penalties on anyone inside their country for merely using encrypted communications. In some countries they might even shoot you for that. But if you live in that kind of country, perhaps you need PGP even more.

5.8 Philip Zimmermann's Legal Situation

At the time of this writing, I am the target of a US Customs criminal investigation in the Northern District of California. A criminal investigation is not a civil lawsuit. Civil lawsuits do not involve prison terms. My defense attorney has been told by the Assistant US Attorney that the area of law of interest to the investigation has to do with the export controls on encryption software. The federal mandatory sentencing guidelines for this offense are 41 to 51 months in a federal prison. US Customs appears to be taking the position that electronic domestic publication of encryption software is the same as exporting it. The prosecutor has issued a number of federal grand jury subpoenas. It may be months before a decision is reached on whether to seek indictment. This situation may change at any time, so this description may be out of date by the time you read it. Watch the news for further developments. If I am indicted and this goes to trial, it will be a major test case.

I have a legal defense fund set up for this case. So far, no other organization is doing the fundraising for me, so I am depending on people like you to contribute directly to this cause. If you care about the future of your civil liberties in the information age, then perhaps you will care about this case. The legal fees are expensive, the meter is running, and I need your help. The fund is run by my lead defense attorney, Phil Dubois, here in Boulder. Please send your contributions to:

```
Philip L. Dubois, Lawyer
2305 Broadway
Boulder, Colorado 80304 USA
Phone (303) 444-3885
E-mail:  dubois@csn.org
```

You can also phone in your donation and put it on Mastercard or Visa. If you want to be really cool, you can use Internet E-mail to send in your contribution, encrypting your message with PGP so that no one can intercept your credit card number. Include in your E-mail message your Mastercard or Visa number, expiration date, name on the card, and amount of donation. Then sign it with your own key and encrypt it with Phil Dubois's public key (his key is included in the standard PGP distribution package, in the *keys.asc* file). Put a note on the subject line that this is a donation to my legal defense fund, so that Mr. Dubois will decrypt it promptly. Please don't send a lot of casual encrypted E-mail to him – I'd rather he use his valuable time to work on my case.

If you want to read some press stories to find out why this is an important case, see the following references:

```
1)  William Bulkeley, "Cipher Probe", Wall Street Journal, Thursday
    28 April 1994, front page.
2)  John Cary, "Spy vs. Computer Nerd:  The Fight Over Data
    Security", Business Week, 4 Oct 1993, page 43.
3)  Jon Erickson, "Cryptography Fires Up the Feds", Dr. Dobb's
    Journal, December 1993, page 6.
4)  John Markoff, "Federal Inquiry on Software Examines Privacy
    Programs", New York Times, Tuesday 21 Sep 1993, page C1.
5)  Kurt Kleiner, "Punks and Privacy", Mother Jones Magazine,
```

Jan/Feb 1994, page 17.

6) Steven Levy, "Battle of the Clipper Chip", New York Times Magazine, Sunday 12 Jun 1994, page 44.

7) Steven Levy, "Crypto Rebels", WIRED, May/Jun 1993, page 54.

8) John Markoff, "Cyberspace Under Lock and Key", New York Times, Sunday 13 Feb 1994.

9) Philip Elmer-DeWitt, "Who Should Keep the Keys", Time, 14 Mar 1994, page 90.

There are a great many other articles on PGP from around the world. I'm keeping a scrapbook.

6 Other sources of information on PGP

6.1 Where to get a commercial version of PGP

To get a fully licensed version of PGP for use in the USA or Canada, contact:

```
ViaCrypt
9033 North 24th Avenue, Suite 7
Phoenix, Arizona 85021  USA
Phone: (602) 944-0773, or (800) 536-2664
Fax: (602) 943-2601
E-mail: viacrypt@acm.org
```

ViaCrypt has a version of PGP for MSDOS, and a number of Unix platforms. They also have a Windows shell version, and other versions are under development, including Macintosh. If you have a need to use PGP in a commercial or Government setting, and ViaCrypt has a version of PGP for your hardware platform, you should get ViaCrypt PGP.

ViaCrypt has obtained all the necessary licenses from PKP, Ascom-Tech AG, and Philip Zimmermann to sell PGP for use in commercial or government environments. ViaCrypt PGP is every bit as secure as the freeware PGP, and is entirely compatible in both directions with the freeware version of PGP. ViaCrypt PGP is the perfect way to get a fully licensed version of PGP into your corporate environment.

If you work in a large company and you are a fan of PGP, I urge you to try to persuade your company to buy lots of copies of PGP from ViaCrypt. Not just because that will earn royalties for me. If ViaCrypt can make PGP a commercial success, it will go a long way toward cementing PGP's political future as an unstoppable standard for E-mail encryption in the corporate world. The corporate world is where the money is, and that affects public policy like nothing else. And that includes Government policy to suppress strong cryptography.

6.2 Reporting PGP bugs

Bugs in PGP should be reported via E-mail to MIT, the official distribution site of PGP. The E-mail address for bug reports is pgp-bugs@mit.edu. MIT will forward a copy of your bug report to me. When you report bugs, be sure to specify what machine and operating system you are using and what version of PGP you have, and provide enough detail to reproduce the problem. It would also be a good idea to find out if you have the latest version of PGP, in case the bug has already been fixed. Also, it's a good idea to make sure it really is a bug before you report it. RTFM.

6.3 Fan mail, updates, and news

After all this work I have to admit I wouldn't mind getting some fan mail for PGP, to gauge its popularity. Let me know what you think about it and how many of your friends use it. Bug reports and suggestions for enhancing PGP are welcome, too. Perhaps a future PGP release will reflect your suggestions.

This project has not been funded and the project has nearly eaten me alive. This means you usually won't get a reply to your mail, unless you only need a short written reply and you include a stamped self-addressed envelope. But I often do reply to E-mail. Please keep it in English, as my foreign language skills are weak. If you call and I'm not in, it's best to just try again later. I usually don't return long distance phone calls, unless you leave a message that I can call you collect, and even then I might not return your call. If you need any significant amount of my time, I am available on a paid consulting basis, and I always return those calls.

The most inconvenient mail I get is for some well-intentioned person to send me a few dollars asking me for a copy of PGP. I don't send it to them because I'd rather avoid any legal problems with PKP. Or worse, sometimes these requests are from foreign countries, and I would be risking a violation of US cryptographic export control laws. Even if there were no legal hassles involved in sending PGP to them, they usually don't send enough money to make it worth my time. I'm just not set up as a low cost low volume mail order business. I can't just ignore the request and keep the money, because they probably regard the money as a fee for me to fulfill their request. If I return the money, I might have to get in my car and drive down to the post office and buy some postage stamps, because these requests rarely include a stamped self-addressed envelope. And I have to take the time to write a polite reply that I can't do it. If I postpone the reply and set the letter down on my desk, it might be buried within minutes and won't see the light of day again for months. Multiply these minor inconveniences by the number of requests I get, and you can see the problem. Isn't it enough that the software is free? It would be nicer if people could try to get PGP from any of the myriad other sources. If you don't have a modem, ask a friend to get it for you. If you can't find it yourself, I don't mind answering a quick phone call.

If anyone wants to volunteer to improve PGP, please let me know. It could certainly use some more work. Some features were deferred to get it out the door. A number of PGP users have since donated their time to port PGP to Unix on Sun SPARCstations, to Ultrix, to VAX/VMS, to OS/2, to the Amiga, and to the Atari ST. Perhaps you can help port it to some new environments. But please let me know if you plan to port or add enhancements to PGP, to avoid duplication of effort, and to avoid starting with an obsolete version of the source code.

Because so many foreign language translations of PGP have been produced, most of them are not distributed with the regular PGP release package because it would require too much disk space. Separate language translation "kits" are available from a number of independent sources, and are sometimes available separately from the same distribution centers that carry the regular PGP release software. These kits include translated versions of the file LANGUAGE.TXT, PGP.HLP, and the PGP User's Guide. If you want to produce a translation for your own native language, contact me first to get the latest information and standard guidelines, and to find out if it's been translated to your language already. To find out where to get a foreign language kit for your language, you might check on the Internet newsgroups, or get it from Mike Johnson (mpj@csn.org).

If you have access to the Internet, watch for announcements of new releases of PGP on the Internet newsgroups *sci.crypt* and PGP's own newsgroup, *alt.security.pgp*. If you want to know where to get PGP, MIT is the primary FTP distribution site (net-dist.mit.edu). Or ask Mike Johnson (mpj@csn.org) for a list of Internet FTP sites and BBS phone numbers.

6.4 Computer-related political groups

PGP is a very political piece of software. It seems appropriate to mention here some computer-related activist groups. Full details on these groups, and how to join them, is provided in a separate document file in the PGP release package.

The Electronic Privacy Information Center (EPIC) is a public interest research center in Washington, DC. It was established in 1994 to focus public attention on emerging privacy issues relating to the National Information Infrastructure, such as the Clipper Chip, the Digital Telephony proposal, medical record privacy, and the sale of consumer data. EPIC is sponsored by the Fund for Constitutional Government and Computer Professionals for Social Responsibility. EPIC publishes the EPIC Alert and EPIC Reports, pursues Freedom of Information Act litigation, and conducts policy research on emerging privacy issues. For more information email info@epic.org, or write EPIC, 666 Pennsylvania Ave., SE, Suite 301, Washington, DC 20003. +1 202 544 9240 (tel), +1 202 547 5482 (fax).

The Electronic Frontier Foundation (EFF) was founded in 1990 to assure freedom of expression in digital media, with a particular emphasis on applying the principles embodied in the US Constitution and the Bill of Rights to computer-based communication. They can be reached in Washington DC, at (202) 347-5400. Internet E-mail address: eff@eff.org.

Computer Professionals For Social Responsibility (CPSR) empowers computer professionals and computer users to advocate for the responsible use of information technology and empowers all who use computer technology to participate in public policy debates on the impacts of computers on society. They can be reached at: (415) 322-3778 in Palo Alto, E-mail address cpsr@csli.stanford.edu.

The League for Programming Freedom (LPF) is a grass-roots organization of professors, students, businessmen, programmers and users dedicated to bringing back the freedom to write programs. They regard patents on computer algorithms as harmful to the US software industry (and so do I!). They can be reached at (617) 433-7071. E-mail address: lpf@uunet.uu.net.

For more details on these groups, see the accompanying document in the PGP release package.

6.5 Recommended readings

Introductory Readings


```
1)  Bruce Schneier, "Applied Cryptography: Protocols, Algorithms, and
    Source Code in C", John Wiley & Sons, 1993
    (This book is a watershed work on the subject.)
2)  Dorothy Denning, "Cryptography and Data Security", Addison-Wesley,
    Reading, MA 1982
3)  Dorothy Denning, "Protecting Public Keys and Signature Keys",
    IEEE Computer, Feb 1983
4)  Martin E. Hellman, "The Mathematics of Public-Key Cryptography,"
    Scientific American, Aug 1979
5)  Steven Levy, "Crypto Rebels", WIRED, May/Jun 1993, page 54.
    (A "must-read" article on PGP and other related topics.)
6)  Steven Levy, "Battle of the Clipper Chip", New York Times
    Magazine, Sunday 12 Jun 1994, page 44. (Great article, great
    photos.)
7)  William Bulkeley, "Cipher Probe", Wall Street Journal, 28 April
    1994, front page.  (An article on PGP and Zimmermann.)
```


Other Readings


```
8)  Ronald Rivest, "The MD5 Message Digest Algorithm", MIT Laboratory
    for Computer Science, 1991
9)  Xuejia Lai, "On the Design and Security of Block Ciphers",
    ETH Series on Information Processing (Ed. J. L. Massey),
    Vol. 1, Hartung-Gorre Verlag, Konstanz, Switzerland, 1992
10) Philip Zimmermann, "A Proposed Standard Format for RSA
    Cryptosystems", Advances in Computer Security, Vol III, edited by
    Rein Turn, Artech House, 1988
11) Paul Wallich, "Electronic Envelopes", Scientific American, Feb
    1993, page 30.  (An article on PGP)
12) William Stallings, "Pretty Good Privacy", BYTE, July 1994, page
    193
13) Philip Zimmermann, "The Official PGP User's Guide", MIT Press,
    1994 (in press)
14) Philip Zimmermann, "PGP Source Code and Internals", MIT Press,
    1994 (in press)
```

6.6 To contact the author

Philip Zimmermann may be reached at:


```
Boulder Software Engineering
3021 Eleventh Street
Boulder, Colorado 80304  USA
Internet:  prz@acm.org
Phone (303) 541-0140 (voice)  (10:00am - 7:00pm Mountain Time)
Fax available, if you arrange it via voice line.
```


A where to get PGP

The following describes how to get the freeware public key cryptographic software PGP (Pretty Good Privacy) from an anonymous FTP site on Internet, or from other sources.

PGP has become a worldwide de facto standard for E-mail encryption. PGP has sophisticated key management, an RSA/conventional hybrid encryption scheme, message digests for digital signatures, data compression before encryption, and good ergonomic design. PGP is well featured and fast, and has excellent user documentation. Source code is free.

The Massachusetts Institute of Technology is the distributor of PGP version 2.6, for distribution in the USA only. It is available from *net-dist.mit.edu*, a controlled FTP site that has restrictions and limitations, similar to those used by RSA Data Security, Inc., to comply with export control requirements. The software resides in the directory /pub/PGP.

A reminder: Set mode to binary or image when doing an FTP transfer. And when doing a kermit download to your PC, specify 8-bit binary mode at both ends.

There are two compressed archive files in the standard release, with the file name derived from the release version number. For PGP version 2.6.2, you must get pgp262.zip which contains the MSDOS binary executable and the PGP User's Guide, and you can optionally get pgp262s.zip which contains all the source code. These files can be decompressed with the MSDOS shareware archive decompression utility PKUNZIP.EXE, version 1.10 or later. For Unix users who lack an implementation of UNZIP, the source code can also be found in the compressed tar file pgp262s.tar.Z.

If you don't have any local BBS phone numbers handy, here is a BBS you might try. The Catacombs BBS, operated by Mike Johnson in Longmont, Colorado, has PGP available for download by people in the US or Canada only. The BBS phone number is 303-772-1062. Mike Johnson's voice phone number is 303 772-1773, and his E-mail address is mpj@csn.org. Mike also has PGP available on an Internet FTP site for users in the US or Canada only; the site name is csn.org, in directory /mpj/, and you must read the README.MPJ file to get it.

To get a fully licensed version of PGP for use in the USA or Canada, contact ViaCrypt in Phoenix, Arizona. Their phone number is 602-944-0773. ViaCrypt has obtained all the necessary licenses from PKP, Ascom-Tech AG, and Philip Zimmermann to sell PGP for use in commercial or Government environments. ViaCrypt PGP is every bit as secure as the freeware PGP, and is entirely compatible in both directions with the freeware version of PGP. ViaCrypt PGP is the perfect way to get a fully licensed version of PGP into your corporate or Government environment.

Here are a few people and their E-mail addresses or phone numbers you can contact in some countries to get information on local PGP availability for versions earlier than 2.5:

```
Peter Gutmann                 Hugh Kennedy
pgut1@cs.aukuni.ac.nz         70042.710@compuserve.com
New Zealand                   Germany

Branko Lankester              Miguel Angel Gallardo
branko@hacktic.nl             gallardo@batman.fi.upm.es
```

+31 2159 42242
The Netherlands

(341) 474 38 09
Spain

Hugh Miller
hmiller@lucpul.it.luc.edu
(312) 508-2727
USA

Colin Plumb
colin@nyx.cs.du.edu
Toronto, Ontario, Canada

Jean-loup Gailly
jloup@chorus.fr
France