Phil's Pretty Good Software
Presents

===
PGP
===

Pretty Good Privacy
Public Key Encryption for the Masses

PGP User's Guide
Volume II: Special Topics

----

by Philip Zimmermann
Revised 1 Sep 92

PGP Version 2.0 - 1 Sep 92
Software Written by
Philip Zimmermann
with
Hal Finney, Branko Lankester, and Peter Gutmann

Synopsis: PGP 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
========

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.

Special Topics

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.

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.

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.

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.

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.

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.

Making a Message For Her Eyes Only

To specify that the recipient's decrypted plaintext will be shown ONLY on her screen and cannot 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.

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 to 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".

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 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.

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]
```

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.

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.

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.

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.

Setting Configuration Parameters: CONFIG.TXT

PGP has a number of user-settable parameters that can be defined in a special 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.

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.

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

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.

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.

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.

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 "canonical 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.

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

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".

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 tend to use PGP mostly for E-mail, it may be a good idea to enable this parameter.

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

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.

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

KEEPBINARY - Keep Binary Ciphertext Files After Decrypting

Default setting: KEEPBINARY = on

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.

VERBOSE - Enable Verbose Mode

Default setting: VERBOSE = off

The configuration parameter VERBOSE enables "verbose" diagnostic messages during PGP's operation, which is mainly useful for debugging PGP. Otherwise, there is not much use for it.

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.

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.

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.

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.

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.

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:

```
PAGER = "more"
```

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".

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.

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 Colorado:    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
```

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.

A remedy for this could involve some trustworthy Certifying Authority or notary that would create notarized signatures with a trustworthy timestamp. This might 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. A public key certificate could be signed by the notary, and the trusted timestamp in the notary's signature would have some legal significance. The notary could enter the signed certificate into a special certificate log controlled by the notary. Anyone can read this log.

The notary could also sign other people's signatures, creating a signature certificate of a signature certificate. This would serve as a witness to the signature the same way real notaries do now with paper. Again, the notary could enter the detached signature certificate (without the actual whole document that was signed) into a log controlled by the notary. The notary's signature would have a trusted timestamp, which might have greater credibility than the timestamp in the original signature. A signature becomes "legal" if it is signed and logged by the notary.

This problem of certifying signatures with notaries and trusted timestamps warrants further discussion. This can of worms will not be fully covered here now. There is a good treatment of this topic in Denning's 1983 article in IEEE Computer (see references). There is much more detail to be worked out in these various certifying schemes. This will develop further as PGP usage increases and other public key products develop their own certifying schemes.

A Peek Under the Hood

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

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.

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.

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) is a good algorithm for most commercial applications. However, the Government does not trust the DES to protect its own classified data. Perhaps this is because the DES key length is 56 bits, short enough for a brute force attack with a special purpose machine built from massive numbers of DES chips. Also, Biham and Shamir have had some success recently on attacking the full 16-round DES.

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). A future version of PGP may support the DES as an option, if enough users ask for it. But I suspect IDEA is better than DES.

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 preliminary evidence suggests that IDEA may be 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. 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.

A famous hockey player once said, "I try to skate to where I think the puck will be." A lot of people are starting to feel that the days are numbered for the DES. I'm skating toward IDEA.

It is not ergonomically practical to use pure RSA with large keys to encrypt and decrypt long messages. Absolutely no one does it that way in the real world. But perhaps you 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. Many people less experienced in cryptography mistakenly believe that RSA is intrinsically stronger than any conventional cipher. It's not. RSA can be made weak by using weak keys, and conventional ciphers can be made strong by choosing good algorithms. It's usually difficult to tell exactly how strong a good conventional cipher is, without actually cracking it. A really good conventional cipher might possibly be harder to crack than even a "military grade" RSA key. 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.

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.

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."

Compatibility with Previous Versions of PGP

I'm sorry, this version of PGP is not compatible with PGP version 1.0. If you have keys generated with version 1.0, you will have to generate new keys to use with this version. This version of PGP uses all new algorithms for conventional cryptography, compression, and message digests, as well as using a much better approach to key management. There were just too many changes to make it compatible with the old format messages, signatures, and keys. Perhaps we could have provided a special conversion utility to convert old keys into new keys, but we were all tired and wanted to get the new release out the door. Besides, converting the old keys into new keys would probably create more problems than it would solve, because we have changed to a new recommended uniform style for the user ID that includes the full name and E-mail address in a particular syntax.

We made some effort to design the internal data structures of this version of PGP to be adaptable to future changes, so that hopefully you will not be required to discard and regenerate your keys in future versions.

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.

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.

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 "How to Protect Public Keys from Tampering" 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.

"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.

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.

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.

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?

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 only running PGP on an isolated single-user system under your direct physical control. That's what I do, and that's what I recommend.

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.

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 cryptanalysis, which has 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.

Legal Issues

Trademarks, Copyrights, and Warranties

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.

Patent Rights on the Algorithms

When I first released PGP, I half-expected to encounter some form of legal harassment from the Government. Indeed, there has been legal harrassment, but it hasn't come from the Government-- it has come from a private corporation.

The RSA public key cryptosystem was developed at MIT with Federal funding from grants from the National Science Foundation and the Navy. It is patented by MIT (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. The author of this software implementation of the RSA algorithm is providing this implementation for educational use only. Licensing this algorithm from PKP is the responsibility of you, the user, not Philip Zimmermann, the author of this software implementation. The author assumes no liability for any patent infringement that may result from the unlicensed use by the user of the underlying RSA algorithm used in this software. Foreign users should note that the RSA patent does not apply outside the US, and there is no RSA patent in any other country. Federal agencies may use it because the Government paid for the development of RSA.

Unfortunately, PKP is not offering any licensing of their RSA patent to end users of PGP. This essentially makes PGP contraband in the USA. Jim Bidzos, president of PKP, threatened to take legal action against me unless I stop distributing PGP, until they can devise a licensing scheme for it. I agreed to this, since PGP is already in wide circulation and waiting a while for a licensing arrangement from PKP seemed reasonable. Mr. Bidzos assured me (he even used the word "promise") several times since the initial 5 June 91 release of PGP that they were working on a licensing scheme for PGP. Apparently, my release of PGP helped provide the impetus for them to offer some sort of a freeware-style license for noncommercial use of the RSA algorithm. However, in December 1991 Mr. Bidzos said he had no plans to ever license the RSA algorithm to PGP users, and denied ever implying that he would. Meanwhile, I have continued to refrain from distributing PGP, although I've recently updated the PGP User's Guide, and have provided a lot of design guidance for these new revisions of PGP.

I wrote my PGP software from scratch, with my own implementation of the RSA algorithm. I didn't steal any software from PKP. Before publishing PGP, 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. However, it is a well known axiom in the US legal system that regardless of the law, he with the most money and lawyers prevails, if not by actually winning then by crushing the little guy with legal expenses.

Not only did PKP acquire the exclusive patent rights for the RSA cryptosystem, which was developed with your tax dollars, but they also somehow acquired the exclusive rights to three other patents covering rival public key schemes invented by others, also developed with your tax dollars. This essentially gives one company 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. And you thought patent law was designed to encourage innovation! PKP does not actually develop any software-- they don't even have an engineering department-- they are essentially a litigation company.

Public key cryptography is destined to become a crucial technology in the protection of our civil liberties and privacy in our increasingly connected society. Why should the Government try to limit access to this key technology, when a single monopoly can do it for them?

It appears certain that there will be future releases of PGP, regardless of the outcome of licensing problems with Public Key Partners. If PKP does not license PGP, then future releases of PGP might not come from me. There are countless fans of PGP outside the US, and many of them are software engineers who want to improve PGP and promote it, regardless of what I do. The second release of PGP was a joint effort of an international team of software engineers, implementing enhancements to the original PGP with design guidance from me. It is being released by Peter Gutmann in New Zealand, out of reach of US patent law. It is being released only in Europe and New Zealand, but it may spontaneously spread to the USA without any 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 patent number is PCT/CH91/00117. International patents are pending. IDEA(tm) is a trademark of Ascom-Tech AG. There is no license fee required for noncommercial use. Commercial users may obtain licensing details from Dieter Profos, Ascom Tech AG, Solothurn Lab, Postfach 151, 4502 Solothurn, Switzerland, Tel +41 65 242885, Fax +41 65 235761.

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 ZIP algorithm, but you're welcome to check into that question yourself. If there are any obscure patent claims that apply to ZIP, then sorry, you'll have to take care of the patent licensing, not me.

All this patent stuff reminds me of a Peanuts cartoon I saw in the newspaper where Lucy showed Charlie Brown a fallen autumn leaf and said "This is the first leaf to fall this year." Charlie Brown said, "How do you know that? Leaves have been falling for weeks." Lucy replied, "I had this one notarized."

Licensing and Distribution

In the USA PKP controls, through US patent law, the licensing of the RSA algorithm. But I have no objection to anyone freely using or distributing my PGP software, without payment of fees to me. You must keep the copyright notices on PGP and keep this documentation with it. However, if you live in the USA, this may not satisfy any legal obligations you may have to PKP for using the RSA algorithm as mentioned above.

In fact, if you live in the USA, and you are not a Federal agency, you shouldn't actually run PGP on your computer, because Public Key Partners wants to forbid you from running my software. PGP is contraband.

Of course, I can't give any assurances, but my guess is that it seems unlikely that PKP would waste their time pursuing PGP end users for patent infringement. There are just too many PGP users to go after. And why would they single you out? But I certainly wouldn't want to imply that you do anything improper-- if PKP were offering licenses, I would urge you to obtain one. But since they aren't, well, I guess you should just refrain from using PGP if you live in the USA.

PGP is not shareware, it's freeware. Forbidden freeware. Published as a community service. If I sold PGP for money, then I would get sued by PKP for using their RSA algorithm. More importantly, giving PGP away for free will encourage far more people to use it, which hopefully will have a greater social impact. This could lead to widespread awareness and use of the RSA public key cryptosystem, which will probably make more money for PKP in the long run. If only they could see that.

All the source code for PGP is available for free under the "Copyleft" General Public License from the Free Software Foundation (FSF). A copy of the FSF General Public License is included in the source release package of PGP.

Regardless of and perhaps contrary to some provisions of the FSF General Public License, the following terms apply:

1) Written discussions of PGP in magazines or books may include fragments of PGP source code and documentation, without restrictions.

2) Although the FSF General Public License allows non-proprietary derivative products, it prohibits proprietary derivative products. Despite this, I may grant you a special license if you want to derive a proprietary commercial product from some of PGP's parts. There may or may not be a fee depending on what kind of a deal you plan to pursue with PKP. Retaining my copyright notice and attribution might suffice in some cases. Give me a call and we'll discuss it. I'm real easy to please.

Feel free to disseminate the complete PGP release package as widely as possible. Give it to all your friends. If you have access to any electronic Bulletin Boards Systems, please upload the complete PGP executable object release package to as many BBS's as possible. You may disseminate the PGP source release package too, if you've got it. The PGP version 2.0 executable object release package for MSDOS contains the PGP executable software, documentation, sample key rings including my own public key, and signatures for the software and this manual, all in one PKZIP compressed file called PGP20.ZIP. The PGP source release package for MSDOS contains all the C source files in one PKZIP compressed file called PGP20SRC.ZIP.

You may obtain free copies or updates to PGP from thousands of BBS's worldwide or from other public sources such as Internet FTP sites. Don't ask me for a copy directly from me, since I'd rather avoid further legal problems with PKP at this time. I might be able to tell you where you can get it, however.

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 can't count on a reply to your mail, unless you only need a short written reply and you include a stamped self-addressed envelope. But I 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. If you need any significant amount of my time, I am available on a paid consulting basis, and I do 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 can't send it to them because of the legal threats from PKP (or worse-- sometimes these requests are from foreign countries, and I would be risking violating 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 ($50 might be worth my time if I were actually selling this stuff). 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, such as the Apple Macintosh, MS Windows, X windows, or XVT. But please let me know if you plan to port PGP, to avoid duplication of effort, and to avoid starting with an obsolete version of the source code.

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. This may cause backward compatibility problems with this version of PGP. Future releases may provide conversion utilities to convert old keys, but you may have to dispose of old messages created with the old PGP.

Export Controls

The Government has made it illegal in many cases to export good cryptographic technology, and that may include PGP. They regard this kind of software as munitions. This is determined by volatile State Department policies, not fixed laws. I will not export this software out of the US or Canada in cases when it is illegal to do so under US State Department policies, and I assume no responsibility for other people exporting it on their own.

If you live outside the US or Canada, I advise you not to violate US State Department policies by getting PGP from a US source. Since thousands of domestic users got it after its initial publication, it somehow leaked out of the US and spread itself widely abroad, like dandelion seeds blowing in the wind. If PGP has already found its way into your country, then I don't think you're violating US export law if you pick it up from a source outside of the US. And there are no import restrictions on bringing cryptographic technology into the USA.

Some foreign governments impose serious penalties on anyone inside their country using encrypted communications. In some countries they might even shoot you for that.

Recommended Introductory Readings

1) Dorothy Denning, "Cryptography and Data Security", Addison-Wesley, Reading, MA 1982
2) Dorothy Denning, "Protecting Public Keys and Signature Keys", IEEE Computer, Feb 1983
3) Martin E. Hellman, "The Mathematics of Public-Key Cryptography," Scientific American, August 1979
4) Philip Zimmermann, "A Proposed Standard Format for RSA Cryptosystems", IEEE Computer, Sep 1986

Other Readings

5) Ronald Rivest, "The MD5 Message Digest Algorithm", MIT Laboratory for Computer Science, 1991
6) Xuejia Lai, "On the Design and Security of Block Ciphers", Institute for Signal and Information Processing, ETH-Zentrum, Zurich, Switzerland, 1992
7) Xuejia Lai, James L. Massey, Sean Murphy, "Markov Ciphers and Differential Cryptanalysis", Advances in Cryptology- EUROCRYPT'91

To Contact the Author

Philip Zimmermann may be reached at:

Boulder Software Engineering
3021 Eleventh Street
Boulder, Colorado 80304 USA
Phone 303-541-0140 (voice or FAX) (10:00am - 7:00pm Mountain Time)
Internet: prz@sage.cgd.ucar.edu