# PGP-INTRO(7) Guide version

Chris Page

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* :<br><br>PGP-INTRO(7) Guide version | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | Chris Page | August 22, 2024 | |

| REVISION HISTORY | | | |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
| | | | |

# Contents

# Chapter 1

# PGP-INTRO(7) Guide version

## 1.1 Welcome

PGP-INTRO User Manual PGP-INTRO

SECURITY FEATURES AND VULNERABILITIES
By Phil Zimmermann

Converted to AmigaGuide format by Chris Page

"Whatever you do will be insignificant, but it is very
important that you do it." -Mahatma Gandhi.



## 1.2 Why I wrote PGP

Why I wrote PGP

It’s personal. It’s private. And it’s no one’s business but yours. You may be planning a political campaign, discussing your taxes, or having a secret romance. Or you may be communicating with a political dissident in a repressive country. Whatever it is, you don’t want your private electronic mail (e-mail) or confidential documents read by anyone else. There’s nothing wrong with asserting your privacy. Privacy is as apple-pie as the Constitution.

The right to privacy is spread implicitly throughout the Bill of Rights. But when the US Constitution was framed, the Founding Fathers saw no need to explicitly spell out the right to a private conversation. That would have been silly. Two hundred years ago, all conversations were private. If someone else was within earshot, you could just go out behind the barn and have your conversation there. No one could listen in without your knowledge. The right to a private conversation was a natural right, not just in a philosophical sense, but in a law-of-physics sense, given the technology of the time. But with the coming of the information age, starting with the invention of the telephone, all that has changed. Now most of our conversations are conducted electronically. This allows our most intimate conversations to be exposed without our knowledge. Cellular phone calls may be monitored by anyone with a radio. Electronic mail, sent across the Internet, is no more secure than cellular phone calls. E-mail is rapidly replacing postal mail, becoming the norm for everyone, not the novelty it was in the past. And e-mail can be routinely and automatically scanned for interesting keywords, on a large scale, without detection. This is like driftnet fishing.

Perhaps you think your e-mail is legitimate enough that encryption is unwarranted. If you really are a law-abiding citizen with nothing to hide, then why don’t you always send your paper mail on postcards? Why not submit to drug testing on demand? Why require a warrant for police searches of your house? Are you trying to hide something? If you hide your mail inside envelopes, does that mean you must be a subversive or a drug dealer, or maybe a paranoid nut? Do law-abiding citizens have any need to encrypt their e-mail?

What if everyone believed that law-abiding citizens should use postcards for their mail? If a nonconformist tried to assert his privacy by using an envelope for his mail, it would draw suspicion. Perhaps the authorities would open his mail to see what he’s hiding. Fortunately, we don’t live in that kind of world, because everyone protects most of their mail with envelopes. So no one draws suspicion by asserting their privacy with an envelope. There’s safety in numbers. Analogously, it would be nice if everyone routinely used encryption for all their e-mail, innocent or not, so that no one drew suspicion by asserting their e-mail privacy with encryption. Think of it as a form of solidarity. Until now, if the government wanted to violate the privacy of ordinary citizens, they had to expend a certain amount of expense and labor to intercept and steam open and read paper mail. Or they had to listen to and possibly transcribe spoken telephone conversation, at least before automatic voice recognition technology became available. This kind of labor-intensive monitoring was not practical on a large scale. This was only done in important cases when it seemed worthwhile.

Senate Bill 266, a 1991 omnibus anti-crime bill, had an unsettling measure buried in it. If this non-binding resolution had become real law, it would have forced manufacturers of secure communications equipment to insert special "trap doors" in their products, so that the government can read anyone’s encrypted messages. It reads:

> "It is the sense of Congress that providers of electronic communications services and manufacturers of electronic communications service equipment shall ensure that communications systems permit the government to obtain the plain text contents of voice, data, and other communications when appropriately authorized by law."

It was this bill that led me to publish PGP electronically for free that year, shortly before the measure was defeated after rigorous protest from civil libertarians and industry groups. The 1994 Digital Telephony bill mandated that phone companies install remote wiretapping ports into their central office digital switches, creating a new technology infrastructure for "point-and-click" wiretapping, so that federal agents no longer have to go out and attach alligator clips to phone lines. Now they'll be able to sit in their headquarters in Washington and listen in on your phone calls. Of course, the law still requires a court order for a wiretap. But while technology infrastructures can persist for generations, laws and policies can change overnight. Once a communications infrastructure optimized for surveillance becomes entrenched, a shift in political conditions may lead to abuse of this new-found power. Political conditions may shift with the election of a new government, or perhaps more abruptly from the bombing of a Federal building.

A year after the 1994 Digital Telephony bill passed, the FBI disclosed plans to require the phone companies to build into their infrastructure the capacity to simultaneously wiretap one percent of all phone calls in all major US cities. This would represent more than a thousandfold increase over previous levels in the number of phones that could be wiretapped. In previous years, there were only about 1000 court-ordered wiretaps in the US per year, at the federal, state, and local levels combined. It's hard to see how the government could even employ enough judges to sign enough wiretap orders to wiretap 1% of all our phone calls, much less hire enough federal agents to sit and listen to all that traffic in real time. The only plausible way of processing that amount of traffic is a massive Orwellian application of automated voice recognition technology to sift through it all, searching for interesting keywords or searching for a particular speaker's voice. If the government doesn't find the target in the first 1% sample, the wiretaps can be shifted over to a different 1% until the target is found, or until everyone's phone line has been checked for subversive traffic. The FBI says they need this capacity to plan for the future. This plan sparked such outrage that it was defeated in Congress, at least this time around, in 1995. But the mere fact that the FBI even asked for these broad powers is revealing of their agenda. And the defeat of this plan isn't so reassuring when you consider that the 1994 Digital Telephony bill was also defeated the first time it was introduced, in 1993.

Advances in technology will not permit the maintenance of the status quo, as far as privacy is concerned. The status quo is unstable. If we do nothing, new technologies will give the government new automatic surveillance capabilities that Stalin could never have dreamed of. The only way to hold the line on privacy in the information age is strong cryptography.

You don't have to distrust the government to want to use cryptography. Your business can be wiretapped by business rivals, organized crime, or foreign governments. The French government, for example, is notorious for using its signals intelligence apparatus against US companies to help French corporations get a competitive edge. Ironically, US government restrictions on cryptography have weakened US corporate defenses against foreign intelligence and organized crime.

The government knows what a pivotal role cryptography is destined to play in the power relationship with its people. In April 1993, the Clinton administration unveiled a bold new encryption policy initiative, which was under development at National Security Agency (NSA) since the start of the Bush administration. The centerpiece of this initiative is a government-built encryption device, called the "Clipper" chip, containing a new classified NSA encryption algorithm. The government has been trying to encourage private industry to design it into all their secure communication products, like secure phones, secure FAX, etc. AT&T has put Clipper into their secure voice products. The catch: At the time of manufacture, each Clipper chip will be loaded with its own unique key, and the government gets to keep a copy, placed in escrow. Not to worry, though-the government promises that they will use these keys to read your traffic only "when duly authorized by law." Of course, to make Clipper completely effective, the next logical step would be to outlaw other forms of cryptography.

The government initially claimed that using Clipper would be voluntary, that no one would be forced to use it instead of other types of cryptography. But the public reaction against the Clipper chip has been strong, stronger than the government anticipated. The computer industry has monolithically proclaimed its opposition to using Clipper. FBI director Louis Freeh responded to a question in a press conference in 1994 by saying that if Clipper failed to gain public support, and FBI wiretaps were shut out by non-government- controlled cryptography, his office would have no choice but to seek legislative relief. Later, in the aftermath of the Oklahoma City tragedy, Mr. Freeh testified before the Senate Judiciary Committee that public availability of strong cryptography must be curtailed by the government (although no one had suggested that cryptography was used by the bombers).

The Electronic Privacy Information Center (EPIC) obtained some revealing documents under the Freedom of Information Act. In a "briefing document" titled "Encryption: The Threat, Applications and Potential Solutions," and sent to the National Security Council in February 1993, the FBI, NSA and Department of Justice (DOJ) concluded that:

> "Technical solutions, such as they are, will only work if they are incorporated into all encryption products. To ensure that this occurs, legislation mandating the use of Government-approved encryption products or adherence to Government encryption criteria is required."

The government has a track record that does not inspire confidence that they will never abuse our civil liberties. The FBI's COINTELPRO program targeted groups that opposed government policies. They spied on the anti-war movement and the civil rights movement. They wiretapped the phone of Martin Luther King Jr. Nixon had his enemies list. And then there was the Watergate mess. Congress now seems intent on passing laws curtailing our civil liberties on the Internet. At no time in the past century has public distrust of the government been so broadly distributed across the political spectrum, as it is today. If we want to resist this unsettling trend in the government to outlaw cryptography, one measure we can apply is to use cryptography as much as we can now while it is still legal.

When use of strong cryptography becomes popular, it's harder for the government to criminalize it. Thus, using PGP is good for preserving democracy.

If privacy is outlawed, only outlaws will have privacy. Intelligence

agencies have access to good cryptographic technology. So do the big arms and drug traffickers. But ordinary people and grassroots political organizations mostly have not had access to affordable "military grade" public-key cryptographic technology. Until now.

PGP empowers people to take their privacy into their own hands. There's a growing social need for it. That's why I created it.

## 1.3 Encryption Basics

First, some elementary terminology. Suppose you want to send a message to a colleague, whom we'll call Alice, and you don't want anyone but Alice to be able to read it. You can encrypt, or encipher the message, which means scrambling it up in a hopelessly complicated way, rendering it unreadable to anyone except you and Alice. You supply a cryptographic key to encrypt the message, and Alice must use the same key to decipher or decrypt it. At least that's how it works in conventional "secret-key" encryption.

A single key is used for both encryption and decryption. This means that this key must be initially transmitted via secure channels so that both parties can know it before encrypted messages can be sent over insecure channels. This may be inconvenient. If you have a secure channel for exchanging keys, then why do you need cryptography in the first place?

## 1.4 How Public Key Cryptography Works

In public key cryptography, everyone has two related complementary keys, a public key and a private key. Each key unlocks the code that the other key makes. Knowing the public key does not help you deduce the corresponding private key. The public key can be published and widely disseminated across a communications network.

This protocol provides privacy without the need for the same kind of secure channels that conventional secret key encryption requires. Anyone can use a recipient's public key to encrypt a message to that person, and that recipient uses her own corresponding private key to decrypt that message. No one but the recipient can decrypt it, because no one else has access to that private key. Not even the person who encrypted the message with the recipient's public key can decrypt it.

## 1.5 How Your Files and Messages are Encrypted

Because the public key encryption algorithm is much slower than conventional single-key encryption, encryption is better accomplished by using the process described below.

A high-quality fast conventional secret-key encryption algorithm is used to encipher the message. This original unenciphered message is called "plaintext." In a process invisible to the user, a temporary random key, created just for

this one "session," is used to conventionally encipher the plaintext file. Then the recipient's public key is used to encipher this temporary random conventional key. This public-key-enciphered conventional "session" key is sent along with the enciphered text (called "ciphertext") to the recipient.

## 1.6 The PGP Symmetric Algorithms

PGP offers a selection of different secret-key algorithms to encrypt the actual message. By secret key algorithm, we mean a conventional, or symmetric, block cipher that uses the same key to both encrypt and decrypt. The three symmetric block ciphers offered by PGP are CAST, Triple-DES, and IDEA. They are not "home-grown" algorithms. They were all developed by teams of cryptographers with distinguished reputations.

For the cryptographically curious, all three ciphers operate on 64-bit blocks of plaintext and ciphertext. CAST and IDEA have key sizes of 128 bits, while triple-DES uses a 168-bit key. Like Data Encryption Standard (DES), any of these ciphers can be used in cipher feedback (CFB) and cipher block chaining (CBC) modes. PGP uses them in 64-bit CFB mode. I included the CAST encryption algorithm in PGP because it shows promise as a good block cipher with a 128-bit key size, it's very fast, and it's free. Its name is derived from the initials of its designers, Carlisle Adams and Stafford Tavares of Northern Telecom (Nortel). Nortel has applied for a patent for CAST, but they have made a commitment in writing to make CAST available to anyone on a royalty-free basis. CAST appears to exceptionally well-designed, by people with good reputations in the field. The design is based on a very formal approach, with a number of formally provable assertions that give good reasons to believe that it probably requires key exhaustion to break its 128-bit key. CAST has no weak or semi-weak keys. There are strong arguments that CAST is completely immune to both linear and differential cryptanalysis, the two most powerful forms of cryptanalysis in the published literature, both of which have been effective in cracking DES. While CAST is too new to have developed a long track record, its formal design and the good reputations of its designers will undoubtedly attract the attentions and attempted cryptanalytic attacks of the rest of the academic cryptographic community. I'm getting nearly the same preliminary gut feeling of confidence from CAST that I got years ago from IDEA, the cipher I selected for use in earlier versions of PGP. At that time, IDEA was also too new to have a track record, but it has held up well.

The IDEA (International Data Encryption Algorithm) block cipher is based on the design concept of "mixing operations from different algebraic groups." It was developed at ETH in Zurich by James L. Massey and Xuejia Lai, and published in 1990. Early published papers on the algorithm called it IPES (Improved Proposed Encryption Standard), but they later changed the name to IDEA. So far, IDEA has resisted attack much better than other ciphers such as FEAL, REDOC-II, LOKI, Snefru and Khafre. And IDEA is more resistant than DES to Biham and Shamir's highly successful differential cryptanalysis attack, as well as attacks from linear cryptanalysis. As this 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. Sadly, the biggest obstacle to IDEA's acceptance as a standard has been the fact that Ascom Systec holds a patent on its design, and unlike DES and CAST, IDEA has not been made available to everyone on a royalty-free basis.

As a hedge, PGP includes three-key triple-DES in its repertoire of available

block ciphers. The DES was developed by IBM in the mid-1970s. While it has a good design, its 56-bit key size is too small by today's standards. Triple-DES is very strong, and has been well studied for many years, so it might be a safer bet than the newer ciphers such as CAST and IDEA. Triple-DES is the DES applied three times to the same block of data, using three different keys, except that the second DES operation is run backwards, in decrypt mode. Although triple-DES is much slower than either CAST or IDEA, speed is usually not critical for e-mail applications. While triple-DES uses a key size of 168 bits, it appears to have an effective key strength of at least 112 bits against an attacker with impossibly immense data storage capacity to use in the attack. According to a paper presented by Michael Weiner at Crypto96, any remotely plausible amount of data storage available to the attacker would enable an attack that would require about as much work as breaking a 129-bit key. Triple-DES is not encumbered by any patents.

PGP public keys that were generated by PGP Version 5.0 or later have information embedded in them that tells a sender what block ciphers are understood by the recipient's software, so that the sender's software knows which ciphers can be used to encrypt. DSS/Diffie-Hellman public keys will accept CAST, IDEA, or triple-DES as the block cipher, with CAST as the default selection. At present, for compatibility reasons, RSA keys do not provide this feature. Only the IDEA cipher is used by PGP to send messages to RSA keys, because older versions of PGP only supported RSA and IDEA.

## 1.7 Data Compression

PGP normally compresses the plaintext before encrypting it, because it's too late to compress the plaintext 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's worth it.

Files that are too short to compress, or that just don't compress well, are not compressed by PGP. In addition, the program recognizes files produced by most popular compression programs, such as PKZIP, and does not try to compress a file that has already been compressed.

For the technically curious, the program uses the freeware ZIP compression routines written by Jean-Loup Gailly, Mark Adler, and Richard B. Wales. This ZIP software uses compression algorithms that are functionally equivalent to those used by PKWare's PKZIP 2.x. This ZIP compression software was selected for PGP mainly because it has a really good compression ratio and because it's fast.

## 1.8 About the Random Numbers used as Session Keys

PGP uses a cryptographically strong pseudo-random number generator for creating temporary session keys. If this random seed file does not exist, it is

automatically created and seeded with truly random numbers derived from your random events gathered by the PGP program from the timing of your keystroke and mouse movements.

This generator reseeds the seed file each time it is used, by mixing in new material partially derived from 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 used to key the conventional encryption engine for the random generator.

This random seed file should be 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 try to keep it from falling into the wrong hands. If possible, make the file readable only by you. If this is not possible, do not let other people indiscriminately copy disks from your computer.

## 1.9 How Decryption Works

The decryption process is just the reverse of encryption. The recipient's private key is used to recover the temporary session key, and then that session key is used to run the fast conventional secret-key algorithm to decipher the large ciphertext message.

## 1.10 How Digital Signatures Work

PGP uses digital signatures to provide message authentication. The sender's own private key can be used to encrypt a message digest, thereby "signing" the message. A message digest is a 160-bit or a 128-bit cryptographically strong one-way hash function. It is somewhat analogous to a "checksum" or CRC error checking code, in that it compactly represents the message and is used to detect changes in the message. Unlike a CRC, however, it is believed to be computationally infeasible for an attacker to devise a substitute message that would produce an identical message digest. The message digest gets encrypted by the sender's private key, creating a digital signature of the message.

The recipient (or anyone else) can verify the digital signature by using the sender's public key to decrypt it. This proves that the sender was the true originator of the message, and that the message has not been subsequently altered by anyone else, because the sender alone possesses the private key that made that signature. Forgery of a signed message is not feasible, and the sender cannot later disavow his signature.

## 1.11 About the Message Digest

The message digest is a compact (160-bit, or 128-bit) "distillate" of your message or file checksum. You can also think of it as a "fingerprint" of the message or file. 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 should 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.

The message digest algorithm now used in PGP (Version 5.0 and later) is called SHA, which stands for Secure Hash Algorithm, designed by the NSA for National Institute of Standards and Technology (NIST). SHA is a 160-bit hash algorithm. Some people might regard anything from the NSA with suspicion, because the NSA is in charge of intercepting communications and breaking codes. But keep in mind that the NSA has no interest in forging signatures, and the government would benefit from a good unforgeable digital signature standard that would preclude anyone from repudiating their signatures. That has distinct benefits for law enforcement and intelligence gathering. Also, SHA has been published in the open literature and has been extensively peer reviewed by most of the best cryptographers in the world who specialize in hash functions, and the unanimous opinion is that SHA is extremely well designed. It has some design innovations that overcome all the observed weaknesses in message digest algorithms previously published by academic cryptographers. All new versions of PGP use SHA as the message digest algorithm for creating signatures with the new DSS keys that comply with the NIST Digital Signature Standard. For compatibility reasons, new versions of PGP still use MD5 for RSA signatures, because older versions of PGP used MD5 for RSA signatures.

The message digest algorithm used by older versions of PGP is the MD5 Message Digest Algorithm, placed in the public domain by RSA Data Security, Inc. MD5 is a 128-bit hash algorithm. In 1996, MD5 was all but broken by Hans Dobbertin, a German cryptographer. While MD5 was not completely broken at that time, it was discovered to have such serious weaknesses that no one should keep using it to generate signatures. Further work in this area might completely break it, thus allowing signatures to be forged. If you don't want to someday find your PGP digital signature on a forged confession, you might be well advised to migrate to the new PGP DSS keys as your preferred method for making digital signatures, because DSS uses SHA as its secure hash algorithm.

## 1.12 How to Protect Public Keys from Tampering

In a public key cryptosystem, you don't have to protect public keys from exposure. In fact, it's better if they are widely disseminated. But it's important to protect public keys from tampering, to make sure that a public key really belongs to whom it appears to belong to. This may be the most important vulnerability of a public key cryptosystem. Let's first look at a potential disaster, then describe how to safely avoid it with PGP. Suppose you want to send a private message to Alice. You download Alice's public key certificate from an electronic bulletin board system (BBS). You encrypt your letter to

Alice with this public key and send it to her through the BBS's e-mail facility.

Unfortunately, unbeknownst to you or Alice, another user named Charlie has infiltrated the BBS and generated a public key of his own with Alice's user ID attached to it. He covertly substitutes his bogus key in place of Alice's real public key. You unwittingly use this bogus key belonging to Charlie instead of Alice's public key. All looks normal because this bogus key has Alice's user ID. Now Charlie can decipher the message intended for Alice because he has the matching private key. He may even re-encrypt the deciphered message with Alice's real public key and send it on to her so that no one suspects any wrongdoing. Furthermore, he can even make apparently good signatures from Alice with this private key because everyone will use the bogus public key to check Alice's signatures.

The only way to prevent this disaster is to prevent anyone from tampering with public keys. If you got Alice's public key directly from Alice, this is no problem. But that may be difficult if Alice is a thousand miles away, or is currently unreachable.

Perhaps you could get Alice's public key from a mutually trusted friend David, who knows he has a good copy of Alice's public key. David could sign Alice's public key, vouching for the integrity of Alice's public key. David would create this signature with his own private key.

This would create a signed public key certificate, and would show that Alice's key had not been tampered with. This requires that you have a known good copy of David's public key to check his signature. Perhaps David could provide Alice with a signed copy of your public key also. David is thus serving as an "Introducer" between you and Alice.

This signed public key certificate for Alice could be uploaded by David or Alice to the BBS, and you could download it later. You could then check the signature via David's public key and thus be assured that this is really Alice's public key. No impostor can fool you into accepting his own bogus key as Alice's because no one else can forge signatures made by David.

A widely trusted person could even specialize in providing this service of "introducing" users to each other by providing signatures for their public key certificates. This trusted person could be regarded as a "Certifying Authority." Any public key certificates bearing the Certifying Authority's signature could be trusted as truly belonging to whom they appear to belong to. All users who wanted to participate would need a known good copy of just the Certifying Authority's public key, so that the Certifying Authority's signatures could be verified. In some cases, the Certifying Authority may also act as a key server, allowing users on a network to look up public keys by asking the key server, but there is no reason why a key server must also certify keys.

A trusted centralized Certifying Authority is especially appropriate for large impersonal centrally controlled corporate or government institutions. Some institutional environments use hierarchies of Certifying Authorities. For more decentralized environments, allowing all users to act as trusted introducers for their friends would probably work better than a centralized key certification authority.

One of the attractive features of PGP is that it can operate equally well in a centralized environment with a Certifying Authority or a more decentralized environment where individuals exchange personal keys. This whole business of

protecting public keys from tampering is the single most difficult problem in practical public key applications. It is the "Achilles heel" of public key cryptography, and a lot of software complexity is tied up in solving this one problem. You should use a public key only after you are sure that it is a good public key that has not been tampered with, and that it actually belongs to the person with whom it purports to be associated. You can be sure of this if you got this public key certificate directly from its owner, or if it bears the signature of someone else that you trust, from whom you already have a good public key. Also, the user ID should have the full name of the key's owner, not just her first name. No matter how tempted you are, you should never give in to expediency and trust a public key you downloaded from a bulletin board, unless it is signed by someone you trust. That uncertified public key could have been tampered with by anyone, maybe even by the system administrator of the bulletin board.

If you are asked to sign someone else's public key certificate, make certain that it really belongs to that person named in the user ID of that public key certificate. This is because your signature on her public key certificate is a promise by you that this public key really belongs to her. Other people who trust you will accept her public key because it bears your signature. It may be ill-advised to rely on hearsay - don't sign her public key unless you have independent first hand knowledge that it really belongs to her. Preferably, you should sign it only if you got it directly from her.

In order to sign a public key, you must be far more certain of that key's ownership than if you merely want to use that key to encrypt a message. To be convinced of a key's validity enough to use it, certifying signatures from trusted introducers should suffice. But to sign a key yourself, you should require your own independent first-hand knowledge of who owns that key. Perhaps you could call the key's owner on the phone and read the key fingerprint to her, to confirm that the key you have is really her key - and make sure you really are talking to the right person.

Bear in mind that your signature on a public key certificate does not vouch for the integrity of that person, but only vouches for the integrity (the ownership) of that person's public key. You aren't risking your credibility by signing the public key of a sociopath, if you are completely confident that the key really belongs to him. Other people would accept that key as belonging to him because you signed it (assuming they trust you), but they wouldn't trust that key's owner. Trusting a key is not the same as trusting the key's owner.

It would be a good idea to keep your own public key on hand with a collection of certifying signatures attached from a variety of "introducers," in the hopes that most people will trust at least one of the introducers who vouch for the validity of your public key. You could post your key with its attached collection of certifying signatures on various electronic bulletin boards. If you sign someone else's public key, return it to them with your signature so that they can add it to their own collection of credentials for their own public key.

PGP keeps track of which keys on your public keyring are properly certified with signatures from introducers that you trust. All you have to do is tell PGP which people you trust as introducers, and certify their keys yourself with your own ultimately trusted key. PGP can take it from there, automatically validating any other keys that have been signed by your designated introducers. And of course you can directly sign more keys yourself.

Make sure that no one else can tamper with your own public keyring. Checking a newly signed public key certificate must ultimately depend on the integrity of the trusted public keys that are already on your own public keyring. Maintain physical control of your public keyring, preferably on your own personal computer rather than on a remote timesharing system, just as you would do for your private key. This is to protect it from tampering, not from disclosure. Keep a trusted backup copy of your public keyring and your private key on write-protected media.

Since your own trusted public key is used as a final authority to directly or indirectly certify all the other keys on your keyring, it is the most important key to protect from tampering. You may wish to keep a backup copy on a write-protected floppy disk.

PGP generally assumes that you will maintain physical security over your system and your keyrings, as well as your copy of PGP itself. If an intruder can tamper with your disk, then in theory he can tamper with the program itself, rendering moot the safeguards the program may have to detect tampering with keys.

One somewhat complicated way to protect your own whole public keyring from tampering is to sign the whole ring with your own private key. You could do this by making a detached signature certificate of the public keyring.

## 1.13 How Does PGP Keep Track of Which Keys are Valid?

Before you read this section, you should read the previous section on How to Protect Public Keys from Tampering.

PGP keeps track of which keys on your public keyring are properly certified with signatures from introducers that you trust. All you have to do is tell PGP which people you trust as introducers, and certify their keys yourself with your own ultimately trusted key. PGP can take it from there, automatically validating any other keys that have been signed by your designated introducers. And of course you may directly sign more keys yourself.

There are two entirely separate criteria PGP uses to judge a public key's usefulness - don't get them confused:

1. Does the key actually belong to whom it appears to belong? In other words, has it been certified with a trusted signature?

2. Does it belong to someone you can trust to certify other keys? PGP can calculate the answer to the first question. To answer the second question, you must tell PGP explicitly. When you supply the answer to question 2, PGP can then calculate the answer to question 1 for other keys signed by the introducer you designated as trusted.

Keys that have been certified by a trusted introducer are deemed valid by PGP. The keys belonging to trusted introducers must themselves be certified either by you or by other trusted introducers. PGP also allows for the possibility of you having several shades of trust for people to act as introducers. Your trust for a key's owner to act as an introducer does not just reflect your estimation of their personal integrity - it should also reflect how competent you think they are at understanding key management and using good

judgment in signing keys. You can designate a person as untrusted, marginally trusted, or completely trusted to certify other public keys. This trust information is stored on your keyring with their key, but when you tell PGP to copy a key off your keyring, PGP will not copy the trust information along with the key, because your private opinions on trust are regarded as confidential. When PGP is calculating the validity of a public key, it examines the trust level of all the attached certifying signatures. It computes a weighted score of validity e.g. two marginally trusted signatures are deemed as credible as one fully trusted signature. The program's skepticism is adjustable - for example, you may tune PGP to require two fully trusted signatures or three marginally trusted signatures to judge a key as valid.

Your own key is "axiomatically" valid to PGP, needing no introducers signature to prove its validity. PGP knows which public keys are yours, by looking for the corresponding private keys on the private key. PGP also assumes you ultimately trust yourself to certify other keys.

As time goes on, you will accumulate keys from other people whom you may want to designate as trusted introducers. Everyone else will choose their own trusted introducers. And everyone will gradually accumulate and distribute with their key a collection of certifying signatures from other people, with the expectation that anyone receiving it will trust at least one or two of the signatures. This will cause the emergence of a decentralized fault tolerant web of confidence for all public keys.

This unique grass-roots approach contrasts sharply with standard public key management schemes developed by government or other monolithic institutions, such as Internet Privacy Enhanced Mail (PEM), which are based on centralized control and mandatory centralized trust. The standard schemes rely on a hierarchy of Certifying Authorities who dictate who you must trust. The program's decentralized probabilistic method for determining public key legitimacy is the centerpiece of its key management architecture. PGP lets you alone choose who you trust, putting you at the top of your own private certification pyramid. PGP is for people who prefer to pack their own parachutes.

Note that while this decentralized, grass-roots approach is emphasized here, it does not mean that PGP does not perform equally as well in the more hierarchical, centralized public key management schemes. Large corporate users, for example, will probably want a central figure or person who signs all the employees' keys. PGP handles that centralized scenario as a special degenerate case of PGP's more generalized trust model.

## 1.14 How to Protect Private Keys from Disclosure

Protect your own private key and your passphrase very carefully. If your private key is ever compromised, you'd better get the word out quickly to all interested parties before someone else uses it to make signatures in your name. For example, they could use it to sign bogus public key certificates, which could create problems for many people, especially if your signature is widely trusted. And of course, a compromise of your own private key could expose all messages sent to you.

To protect your private key, you can start by always keeping physical control of your private key. Keeping it on your personal computer at home is

OK, or keep it in your notebook computer that you can carry with you. If you must use an office computer that you don't always have physical control of, then keep your public and private keyrings on a write-protected removable floppy disk, and don't leave it behind when you leave the office. It wouldn't be a good idea to allow your private key to reside on a remote time-sharing computer, such as a remote dial-in UNIX system. Someone could eavesdrop on your modem line and capture your passphrase and then obtain your actual private key from the remote system. You should only use your private key on a machine that is under your physical control.

Don't store your passphrase anywhere on the computer that has your private key file. Storing both the private key and the passphrase on the same computer is as dangerous as keeping your PIN in the same wallet as your Automatic Teller Machine bank card. You don't want somebody to get their hands on your disk containing both the passphrase and the private key file. It would be most secure if you just memorize your passphrase and don't store it anywhere but your brain. If you feel you must write down your passphrase, keep it well protected, perhaps even more well protected than the private key file.

And keep backup copies of your private key-remember, you have the only copy of your private key, and losing it will render useless all the copies of your public key that you have spread throughout the world.

The decentralized non-institutional approach PGP supports for management of public keys has its benefits, but unfor tunately this also means we can't rely on a single centralized list of which keys have been compromised. This makes it a bit harder to contain the damage of a private key compromise. You just have to spread the word and hope everyone hears about it.

If the worst case happens - your private key and passphrase are both compromised (hopefully you will find this out somehow) - you will have to issue a "key compromise" certificate. This kind of certificate is used to warn other people to stop using your public key. You can use PGP to create such a certificate by using the Revoke command from the PGPkeys menu. Then you must somehow send this compromise certificate to everyone else on the planet, or at least to all your friends and their friends, et cetera. Their own PGP software will install this key compromise certificate on their public keyrings and will automatically prevent them from accidentally using your public key ever again. You can then generate a new private/public key pair and publish the new public key. You could send out one package containing both your new public key and the key compromise certificate for your old key.

## 1.15 What If You Lose Your Private Key?

Normally, if you want to revoke your own private key, you can use the Revoke command from the PGPkeys menu to issue a revocation certificate, signed with your own private key.

But what can you do if you lose your private key, or if your private key is destroyed? You can't revoke it yourself, because you must use your own private key to revoke it, and you don't have it anymore. You ask each person you signed your key to retire his/her certification. Then anyone attempting to use your key based upon the trust of one of your introducers will know not to trust your public key.

## 1.16 Beware of Snake Oil

When examining a cryptographic software package, the question always remains, why should you trust this product? Even if you examined the source code yourself, not everyone has the cryptographic experience to judge the security. Even if you are an experienced cryptographer, subtle weaknesses in the algorithms could still elude you.

When I was in college in the early seventies, I devised what I believed was a brilliant encryption scheme. A simple pseudorandom number stream was added to the plaintext stream to create ciphertext. This would seemingly thwart any frequency analysis of the ciphertext, and would be uncrackable even to the most resourceful government intelligence agencies. I felt so smug about my achievement.

Years later, I discovered this same scheme in several introductory cryptography texts and tutorial papers. How nice. Other cryptographers had thought of the same scheme. Unfortunately, the scheme was presented as a simple homework assignment on how to use elementary cryptanalytic techniques to trivially crack it. So much for my brilliant scheme.

From this humbling experience I learned how easy it is to fall into a false sense of security when devising an encryption algorithm. Most people don't realize how fiendishly difficult it is to devise an encryption algorithm that can withstand a prolonged and determined attack by a resourceful opponent. Many mainstream software engineers have developed equally naive encryption schemes (often even the very same encryption scheme), and some of them have been incorporated into commercial encryption software packages and sold for good money to thousands of unsuspecting users.

This is like selling automotive seat belts that look good and feel good, but snap open in even the slowest crash test. Depending on them may be worse than not wearing seat belts at all. No one suspects they are bad until a real crash. Depending on weak cryptographic software may cause you to unknowingly place sensitive information at risk. You might not otherwise have done so if you had no cryptographic software at all. Perhaps you may never even discover your data has been compromised.

Sometimes commercial packages use the Federal Data Encryption Standard (DES), a fairly good conventional algorithm recommended by the government for commercial use (but not for classified information, oddly enough-Hmmm). There are several "modes of operation" DES can use, some of them better than others. The government specifically recommends not using the weakest simplest mode for messages, the Electronic Codebook (ECB) mode. But they do recommend the stronger and more complex Cipher Feedback (CFB) or Cipher Block Chaining (CBC) modes.

Unfortunately, most of the commercial encryption packages I've looked at use ECB mode. When I've talked to the authors of a number of these implementations, they say they've never heard of CBC or CFB modes, and didn't know anything about the weaknesses of ECB mode. The very fact that they haven't even learned enough cryptography to know these elementary concepts is not reassuring. And they sometimes manage their DES keys in inappropriate or insecure ways. Also, these same software packages often include a second faster encryption algorithm that can be used instead of the slower DES. The author of the package often thinks his proprietary faster algorithm is as secure as DES, but after questioning him

I usually discover that it's just a variation of my own brilliant scheme from college days. Or maybe he won't even reveal how his proprietary encryption scheme works, but assures me it's a brilliant scheme and I should trust it. I'm sure he believes that his algorithm is brilliant, but how can I know that without seeing it?

In all fairness I must point out that in most cases these terribly weak products do not come from companies that specialize in cryptographic technology.

Even the really good software packages, that use DES in the correct modes of operation, still have problems. Standard DES uses a 56-bit key, which is too small by today's standards, and may now be easily broken by exhaustive key searches on special high-speed machines. The DES has reached the end of its useful life, and so has any software package that relies on it.

There is a company called AccessData (87 East 600 South, Orem, Utah 84058, phone 1-800-658-5199) that sells a package for $185 that cracks the built-in encryption schemes used by WordPerfect, Lotus 1-2-3, MS Excel, Symphony, Quattro Pro, Paradox, MS Word, and PKZIP. It doesn't simply guess passwords - it does real cryptanalysis. Some people buy it when they forget their password for their own files. Law enforcement agencies buy it too, so they can read files they seize. I talked to Eric Thompson, the author, and he said his program only takes a split second to crack them, but he put in some delay loops to slow it down so it doesn't look so easy to the customer.

In the secure telephone arena, your choices look bleak. The leading contender is the STU-III (Secure Telephone Unit), made by Motorola and AT&T for $2000-$3000, and used by the government for classified applications. It has strong cryptography, but requires some sort of special license from the government to buy this strong version. A commercial version of the STU-III is available that is watered down for NSA's convenience, and an export version is available that is even more severely weakened. Then there is the $1200 AT&T Surity 3600, which uses the government's famous Clipper chip for encryption, with keys escrowed with the government for the convenience of wiretappers. Then of course, there are the analog (non-digital) voice scramblers that you can buy from the spywannabe catalogs, that are really useless toys as far as cryptography is concerned, but are sold as "secure" communications products to customers who just don't know any better.

In some ways, cryptography is like pharmaceuticals. Its integrity may be absolutely crucial. Bad penicillin looks the same as good penicillin. You can tell if your spreadsheet software is wrong, but how do you tell if your cryptography package is weak? The ciphertext produced by a weak encryption algorithm looks as good as ciphertext produced by a strong encryption algorithm. There's a lot of snake oil out there. A lot of quack cures. Unlike the patent medicine hucksters of old, these software implementors usually don't even know their stuff is snake oil. They may be good software engineers, but they usually haven't even read any of the academic literature in cryptography. But they think they can write good cryptographic software. And why not? After all, it seems intuitively easy to do so. And their software seems to work okay.

Anyone who thinks they have devised an unbreakable encryption scheme either is an incredibly rare genius or is naive and inexperienced. Unfortunately, I sometimes have to deal with would-be cryptographers who want to make "improvements" to PGP by adding encryption algorithms of their own design.

I remember a conversation with Brian Snow, a highly placed senior

cryptographer with the NSA. He said he would never trust an encryption algorithm designed by someone who had not "earned their bones" by first spending a lot of time cracking codes. That did make a lot of sense. I observed that practically no one in the commercial world of cryptography qualified under this criterion. "Yes," he said with a self assured smile, "And that makes our job at NSA so much easier." A chilling thought. I didn't qualify either.

The government has peddled snake oil too. After World War II, the US sold German Enigma ciphering machines to third world governments. But they didn't tell them that the Allies cracked the Enigma code during the war, a fact that remained classified for many years. Even today many UNIX systems worldwide use the Enigma cipher for file encryption, in part because the government has created legal obstacles against using better algorithms. They even tried to prevent the initial publication of the RSA algorithm in 1977. And they have for many years squashed essentially all commercial efforts to develop effective secure telephones for the general public.

The principal job of the US government's National Security Agency is to gather intelligence, principally by covertly tapping into people's private communications (see James Bamford's book, The Puzzle Palace). The NSA has amassed considerable skill and resources for cracking codes. When people can't get good cryptography to protect themselves, it makes NSA's job much easier. NSA also has the responsibility of approving and recommending encryption algorithms. Some critics charge that this is a conflict of interest, like putting the fox in charge of guarding the hen house. In the 1980s, NSA had been pushing a conventional encryption algorithm that they designed (the COMSEC Endorsement Program), and they won't tell anybody how it works because that's classified. They wanted others to trust it and use it. But any cryptographer can tell you that a well-designed encryption algorithm does not have to be classified to remain secure. Only the keys should need protection. How does anyone else really know if NSA's classified algorithm is secure? It's not that hard for NSA to design an encryption algorithm that only they can crack, if no one else can review the algorithm. And now with the Clipper chip, the NSA is pushing SKIPJACK, another classified cipher they designed. Are they deliberately selling snake oil?

There are three main factors that have undermined the quality of commercial cryptographic software in the US.

- The first is the virtually universal lack of competence of implementors of commercial encryption software (although this is starting to change since the publication of PGP). Every software engineer fancies himself a cryptographer, which has led to the proliferation of really bad crypto software.

- The second is the NSA deliberately and systematically suppressing all the good commercial encryption technology, by legal intimidation and economic pressure. Part of this pressure is brought to bear by stringent export controls on encryption software which, by the economics of software marketing, has the net effect of suppressing domestic encryption software.

- The other principle method of suppression comes from the granting all the software patents for all the public key encryption algorithms to a single company, affording a single choke point to suppress the spread of this technology (although this crypto patent cartel broke up in the fall of 1995).

The net effect of all this is that before PGP was published, there was almost no highly secure general purpose encryption software available in the US.

I'm not as certain about the security of PGP as I once was about my brilliant encryption software from college. If I were, that would be a bad sign. But I don't think PGP contains any glaring weaknesses (although I'm pretty sure it contains bugs). I have selected the best algorithms from the published literature of civilian cryptologic academia. For the most part, they have been individually subject to extensive peer review. I know many of the world's leading cryptographers, and have discussed with some of them many of the cryptographic algorithms and protocols used in PGP. It's well researched, and has been years in the making. And I don't work for the NSA. But you don't have to trust my word on the cryptographic integrity of PGP, because source code is available to facilitate peer review.

And one more point about my commitment to cryptographic quality in PGP: Since I first developed and released PGP for free in 1991, I spent three years under criminal investigation by US Customs for PGP's spread overseas, with risk of criminal prosecution and years of imprisonment (by the way, you didn't see the government getting upset about other cryptographic software - it's PGP that really set them off - what does that tell you about the strength of PGP?). I have earned my reputation on the cryptographic integrity of my products. I will not betray my commitment to our right to privacy, for which I have risked my freedom. I'm not about to allow a product with my name on it to have any secret back doors.

## 1.17 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.

> "If all the personal computers in the world - 260 million - were put to work on a single PGP-encrypted message, it would still take an estimated 12 million times the age of the universe, on average, to break a single message."
> - William Crowell, Deputy Director,
> National Security Agency, March 20, 1997.

### Compromised passphrase and Private Key

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

Here are some recommendations for protecting your passphrase:

1. Don't use obvious passphrases that can be easily guessed, such as

the names of your kids or spouse.

2. Use spaces and a combination of numbers and letters in your passphrase. If you make your passphrase 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 passphrase is so much better than a password. A more sophisticated attacker may have his computer scan a book of famous quotations to find your passphrase.

3. Be creative. Use an easy to remember but hard to guess passphrase; you can easily construct one by using some creatively nonsensical sayings or very obscure literary quotes.

## 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 earlier in this document.

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 keyring. Maintain physical control of both your public keyring and your private key, preferably on your own personal computer rather than on a remote timesharing system. Keep a backup copy of both keyrings.

## 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.

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 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 MS-DOS 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 Passphrase or private 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. You should make an effort to get your copy of PGP directly from Pretty Good Privacy.

There are other ways to check PGP for tampering, using digital signatures. You could use another trusted version of PGP to check the signature on a 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.

Swap Files or Virtual Memory PGP was originally developed for MS-DOS, a primitive operating system by today's standards. But as it was ported to other more complex operating systems, such as Microsoft Windows or the Macintosh OS, a new vulnerability emerged. This vulnerability stems from the fact that these fancier operating systems use a technique called virtual memory.

Virtual memory allows you to run huge programs on your computer that are bigger than the space available in your computer's semiconductor memory chips. This is handy because software has become more and more bloated since graphical user interfaces became the norm, and users started running several large applications at the same time. The operating system uses the hard disk to store portions of your software that aren't being used at the moment. This means that the operating system might, without your knowledge, write out to disk some things that you thought were kept only in main memory. Things like keys, passphrases, or decrypted plaintext. PGP does not keep that kind of sensitive data lying around in memory for longer than necessary, but these is some chance that the operating system could write it out to disk anyway.

The data is written out to some scratchpad area of the disk, known as a swap file. Data is read back in from the swap file as needed, so that only part of your program or data is in physical memory at any one time. All this activity is invisible to the user, who just sees the disk chattering away. Microsoft Windows swaps chunks of memory, called pages, using a Least Recently Used (LRU) page replacement algorithm. This means pages that have not been accessed for the longest period of time are the first ones to be swapped to the disk. This approach suggest that in most cases the risk is fairly low that sensitive data will be swapped out to disk, because PGP doesn't leave it in memory for very long. But we don't make any guarantees.

This swap file may be accessed by anyone who can get physical access to your computer. If you are concerned about this problem, you may be able to solve it by obtaining special software that overwrites your swap file. Another possible cure is to turn off your operating system's virtual memory feature. Microsoft Windows allows for this, and so does the Mac OS. Turning off virtual memory means you might need to have more physical RAM chips installed in order to fit everything in RAM.

## 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 grass-roots political organizations that depend on a largely volunteer staff.

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.

## 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/private 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 the Recommended Introductory Readings section, below). Future enhancements to PGP might have features to easily manage notarized signatures of signatures, with trusted timestamps.

## Exposure on Multi-user Systems

PGP was originally designed for a single-user PC under your direct physical control. If you run PGP at home on your own PC your encrypted files are generally safe, unless someone breaks into your house, steals your PC and convinces you to give them your passphrase (or your passphrase is simple enough to guess).

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 private 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.

## 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. But civilian academia has been intensively attacking it without success since 1978.

Perhaps the government has some classified methods of cracking the IDEA 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 DES in withstanding differential cryptanalysis.

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.

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.

## 1.18 Recommended Introductory & Other readings

Recommended Introductory Readings

- o Bacard Andre, "Computer Privacy Handbook," Peachpit Press, 1995
- o Garfinkel Simson, "Pretty Good Privacy," O'Reilly & Associates, 1995
- o Schneier Bruce, "Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition," John Wiley & Sons, 1996
- o Schneier Bruce, "E-mail Security," John Wiley & Sons, 1995
- o Stallings William, "Protect Your Privacy," Prentice Hall, 1994

Other Readings:

- o Lai Xuejia, "On the Design and Security of Block Ciphers," Institute for Signal and Information Processing, ETH-Zentrum, Zurich, Switzerland, 1992
- o Lai Xuejia, Massey James L., Murphy Sean" Markov Ciphers and Differential Cryptanalysis," Advances in Cryptology - EUROCRYPT'91
- o Rivest Ronald, "The MD5 Message Digest Algorithm," MIT Laboratory for Computer Science, 1991
- o Wallich Paul, "Electronic Envelopes," Scientific American, Feb. 1993, page 30.
- o Zimmermann Philip, "A Proposed Standard Format for RSA Cryptosystems," Advances in Computer Security, Vol. III, edited by Rein Turn, Artech House, 1988 Chapter 6.

## 1.19 About

This document was converted from ansi format man pages, provided by Stefan Zakarias <stef@amitar.com.au> as part of the amiga-pgp51i-bin.lha archive, to Amigaguide format by Chris Page <dasoft@zetnet.co.uk> using GoldED 4.7.2.

While every effort has been taken to ensure that the contents of this document have remained unchanged beyond that necessary to reformat the text, the converter may not be held responsible for errors, ommisions or additions (whether of context, meaning or content) in this version of the document. All character formatting (Bold/ Italic etc) is for presentation alone and is not intended to alter the meaning of the text formatted.

Please do not contact Chris Page with questions concerning the content of this document - I only converted it, not wrote it :)