Phil's Pretty Good Software Presents...

PGPfone

Pretty Good Privacy Phone

Owner's Manual

Version 1.0 beta 7 -- 8 July 1996

Philip R. Zimmermann

Phil's Pretty Good engineering team:

PGPfone for the Apple Macintosh and Windows written mainly by Will Price.

Phil Zimmermann:	Overall application design, cryptographic and key management protocols, call setup negotiation, and, of course, the manual.
Will Price:	Overall application design. He persuaded the rest of the team to abandon the original DOS command-line approach and designed a multithreaded event-driven GUI architecture. Also greatly improved call setup protocols.
Chris Hall:	Did early work on call setup protocols and cryptographic and key management protocols, and did the first port to Windows.
Colin Plumb:	Cryptographic and key management protocols, call setup negotiation, and the fast multiprecision integer math package.
Jeff Sorensen:	Speech compression.
Will Kinney:	Optimization of GSM speech compression code.
Kelly MacInnis:	Early debugging of the Win95 version.
Patrick Juola:	Computational linguistic research for biometric word list.

Table of Contents

Acknowledgments

In addition to the engineering team credited above, PGPfone owes much to other people who have contributed their time and ideas. My own mostly pro-bono legal defense team provided legal advice that was essential for safely (we hope) releasing this product. My legal defense team includes Phil Dubois, Ken Bass, Curt Karnow, Eben Moglen, John Ogilvie, Bob Corn-Revere, and Tom Nolan. Jeff Schiller and Hal Abelson made it possible for PGPfone to be released through the MIT Web site. Zhahai Stewart contributed the early ideas for constructing the biometric authentication word list that Patrick Juola created for us.

The GSM speech compression routines are derived from code that is copyright 1992 by Jutta Degener and Carsten Bormann, Technische Universitaet Berlin. The TripleDES code in PGPfone was provided by Richard Outerbridge. The Blowfish code was derived from code provided by Bruce Schneier. The CAST code was derived from code provided by Northern Telecom.

MetroWerks donated their CodeWarrior C++ development system to facilitate development of the Mac version of PGPfone. Rich Ward at Microsoft donated Microsoft Visual C++ for the Win95 version. IBM donated VisualAge C++ for the OS/2 version (under development).

Introduction

What is PGPfone™?

PGPfone lets you whisper in someone's ear, even if their ear is a thousand miles away.

PGPfone (Pretty Good Privacy Phone) is a software package that turns your desktop or notebook computer into a secure telephone. It uses speech compression and strong cryptographic protocols to give you the ability to have a real-time secure telephone conversation. PGPfone takes your voice from a microphone, then continuously digitizes, compresses and encrypts it and sends it out through a modem to the person at the other end who is also running PGPfone. All cryptographic and speech compression protocols are negotiated dynamically and invisibly, providing a natural user interface similar to using a normal telephone. Public-key protocols are used to negotiate keys without the need for secure channels.

All you need to run PGPfone is:

1) A really reliable fast modem: at least 14.4 Kbps V.32bis (28.8 Kbps V.34 recommended).
2) And either:

   An Apple® Macintosh™ with at least a 25MHz 68LC040 processor (PowerPC recommended), running System 7.1 or above, Thread Manager 2.0.1, ThreadsLib 2.1.2, and Sound Manager 3.0. These are available from Apple's FTP sites. We do not claim that PGPfone runs on a 68030 Mac, but you may find special situations that will work. Also, not all 68040 Macs will run it, depending on whether they have the necessary sound input hardware.

   or

   A multimedia PC running Windows 95 or NT, with at least a 66 MHz 486 CPU (Pentium recommended), sound card, microphone, and speakers or headphones.

For the impatient: If you need to start using PGPfone right this very minute, you may skip on ahead to the section on How to Use PGPfone. But if you don't skip ahead and just keep reading, you'll find some very interesting stuff about why cryptography is so important to your privacy and civil liberties, and why good cryptography is so hard to come by.

For important information about licensing, export controls, patents, copyrights, trademarks, and warranties, see the Legal Issues section later in the manual.

For the technically curious: PGPfone does not need any secure channels for the prior exchange of cryptographic keys before the conversation begins. The two parties negotiate their keys using the Diffie-Hellman key exchange protocol, which reveals nothing useful to a wiretapper, yet allows the

two parties to arrive at a common key that they can use to encrypt and decrypt their voice streams. PGPfone version 1.0 uses biometric signatures (your voice) to authenticate the key exchange, triple-DES, CAST, or Blowfish for encrypting the voice stream, and GSM for the speech compression. More on all that later.

Why We Made PGPfone, and Why You Need it.

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 an illicit affair. Or you may be doing something that you feel shouldn't be illegal, but is. Whatever it is, you don't want your telephone calls to be intercepted or overheard 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, both business and personal, to be exposed without our knowledge. Cellular phone calls may be monitored by anyone with a radio. Electronic mail can be routinely scanned for interesting keywords, on a large scale. This driftnet fishing approach has been readily applicable to email for a long time, but in recent years advances in voice recognition technology have begun to bring similar capabilities to filtering phone calls. Now the government can scan large numbers of phone calls for particular words, or for particular individual's voices. I'm not saying the government actually does this to domestic phone calls today on a large scale as a matter of policy, but they have acquired the technology nonetheless.

In 1991, Senate Bill 266 included a non-binding resolution, which if it had become real law, would have forced manufacturers of secure communications equipment to insert special "trap doors" in their products, so that the government could read anyone's encrypted messages. Before that measure was defeated, I wrote and released Pretty Good Privacy, my email encryption software that uses public-key encryption algorithms. I did it because I wanted cryptography to be made available to the American public before it became illegal to use it. I gave it away for free so that it would achieve wide dispersal, to inoculate the body politic.

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 to your phone calls. Of course, the law still requires a court order for a wiretap. But while technology infrastructures tend to 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. Cryptography strong enough to keep out major governments.

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 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.

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 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."

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). A few months later, Senator Grassley introduced legislation that would outlaw placing cryptographic software on any computer network that might be accessible by a foreigner -- in other words, any computer network. The only exception would be if the software were designed to escrow its keys with the government. At the time of this writing, the fate of Grassley's bill is still pending.

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

Martin Luther King's phone. Nixon had his enemies list. And then there was the Watergate mess. The War on Drugs has given America the world's largest per-capita incarceration rate in the world, a distinction formerly held by South Africa, before we surpassed them during the eighties even when apartheid was in full swing. Recently, we've seen the images and sounds of the Rodney King beatings, Detective Mark Fuhrman's tapes boasting of police abuses, and the disturbing events of the Ruby Ridge case. And now Congress 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 and PGPfone 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. So do defense contractors and some other corporate giants. But ordinary people and grassroots political organizations mostly have not had access to affordable "military grade" public-key cryptographic technology for telephone conversations. Until now.

PGPfone, like the original PGP, empowers people to take their privacy into their own hands. It seems that it is now once again time for direct action, before it becomes illegal to spread this technology. So here is PGPfone.

Beware of Snake Oil

(This section is mostly lifted from the PGP User's Guide)

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" the 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 the 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 the 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 to buy this strong version, and even the strong version has a back door for the NSA. A commercial version of the STU-III is available that is somewhat watered down, and an export version is available that is even more severely weakened for NSA's convenience. 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 spy-wannabe 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 PGPfone 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 PGPfone 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 PGPfone. 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 PGPfone, because source code is available to facilitate peer review.

And one more point about my commitment to cryptographic quality in PGPfone: 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 Feds 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.

How to Use PGPfone

Getting Started

The instructions that follow apply equally in almost every case to both the Windows and Macintosh versions of PGPfone. The screen shots are mostly from the Macintosh version, but the Windows screens are virtually identical. The beginner mode screen for PGPfone during a full-duplex call looks like this on a Macintosh:

Figure 1. PGPfone's minimal front panel

The main window shown above has three modes: beginner, intermediate, and advanced. These correspond basically to how much information you want to have on your screen about the call. When placed into advanced mode from the Window menu, the Windows versions looks like this:

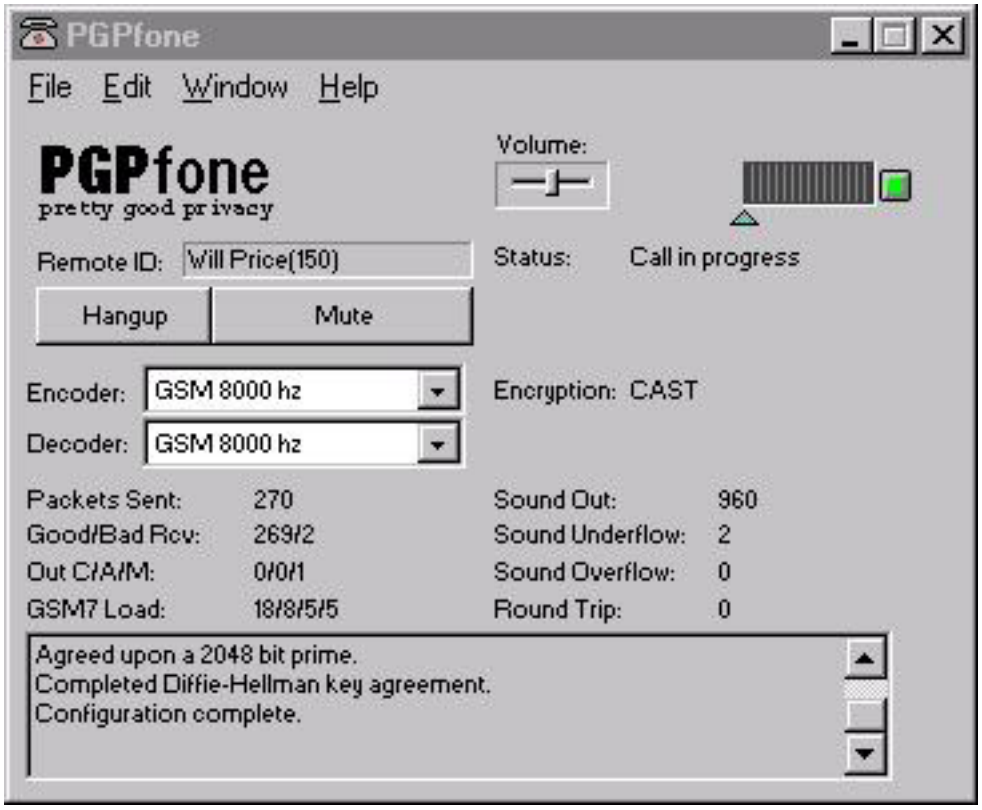

Figure 2a: PGPfone's full front panel for Windows

Whereas the Macintosh versions looks like this:

Figure 2b: PGPfone's full front panel for Mac

Macintosh Installation

To install PGPfone on the Mac, just unpack it from its self-extracting archive, which creates a folder caller PGPfone. In this folder, you should find the PGPfone executable program, the PGPfone Owner's Manual, and a small README file, which contains release notes.

The first time you run PGPfone on a Mac, it will create a "PGPfone Preferences" file which PGPfone places in the Preferences folder in the System Folder.

Windows Installation

To install PGPfone on Windows, unzip it from its archive. In this folder, you should find the SETUP.EXE file. Run this to install PGPfone under Windows 95 and Windows NT. PGPfone does not run under Windows 3.1. When the installation is complete, the PGPfone executable program, PGPfone Owner's Manual, a README file which contains release notes, and several sound files that must remain in the same folder as PGPfone will have been installed in the directory you specify.

Preferences Settings

PGPfone has some user preferences that can be set in some dialog boxes within PGPfone. To get started running PGPfone, you can leave most of these preferences in their default settings.

If you need help understanding any of the controls in the preferences dialog box, note that Balloon Help is available throughout all of PGPfone's dialog boxes on the Macintosh version, and clicking the "Help" button will provide help in the Windows preferences dialog.

Here are some particularly important settings you should set in the user preferences.

Serial/Modem Dialog Box:

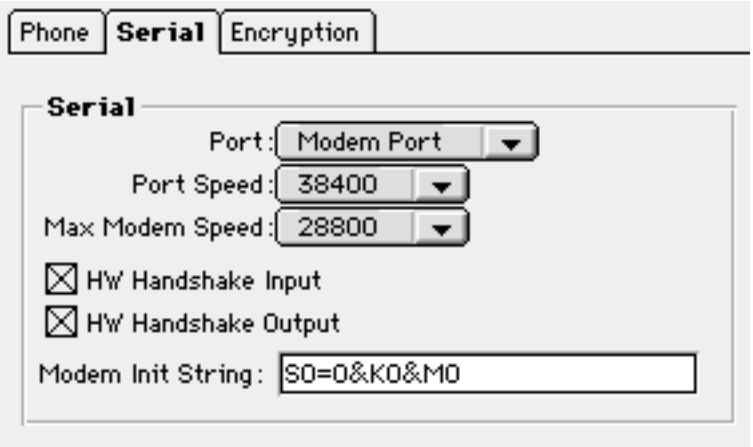

Figure 3: Serial/modem dialog box

Serial Port Selection: Select the right port that your modem is connected to. Most of the time, for most Macs, this will be the modem port, not the printer port. That is the default setting. For Windows machines, the ports are labeled as COM #. Pick the number of port to which the modem you wish to use PGPfone with is connected. If you do not wish to use PGPfone over a modem, you can ignore these preferences.

Max modem speed: Set this to the maximum possible speed of your modem, for example, 28800 baud or 14400 baud. This is not the same as the speed of your serial port which connects to your modem. The serial port speed is usually set to a faster speed than the maximum modem speed.

Port speed: Set this to the port rate used to communicate with your modem. This is not the same as the baud rate of your modem. If you have a 14.4kbps modem, you will want to set this to something greater than or equal to 19200. For a 28.8kbps modem, you should set it to at least 38400. If you have a fast enough computer to handle it, using 57600 as a port rate is usually best in this case.

Some strange modems, like the Global Village PowerPort Bronze 9600 baud modem on some older powerbooks, require the port speed to be the same as the maximum baud rate, instead of the next higher speed up from that speed. You will probably be able to find AT commands in your modem's manual to cure such problems. But you probably shouldn't be trying to run PGPfone on those older Powerbooks anyway.

Modem init string: This must be set up to get the modem to do two things:

1) Not automatically answer the phone line if it rings. Most modems use the command (S0=0) to tell the modem to not answer the phone if it rings.

2) Turn off all error correction and compression, such as V.42bis or MNP/5. If you have even slightly noisy phone lines, PGPfone works much better if you turn off the modem's error correction. Every modem manufacturer uses a different command to turn off error correction, so we can't tell you one command that will always work. Consult your modem's user manual.

   Turning off the modem's error correction features is really important to PGPfone's performance. When error correction is enabled, the modem retransmits data that was received in error by the other modem. This is good for transmitting files, but very bad for transmitting a continuous real-time voice stream. The small delay from the retransmission causes PGPfone's outgoing data buffers (and the operating system's serial output buffers, and the modem's own internal buffers) to get backlogged a little, and each time another error delay occurs, the backlog grows a bit longer. This results in a gradually-increasing latency in how long it takes your voice to reach the other party. It can grow to several *seconds*. This makes your conversations feel like you are talking to an astronaut out near the Moon or beyond.

   Actually, if you are quite certain that you have absolutely noiseless phone lines, you'll get better throughput if you leave error correction turned on, because that forces the modem to run in synchronous mode, without start and stop bits, which saves you about 20% of the transmission overhead. But if there is any line noise whatsoever that would trigger the modem's error correction (which involves retransmitting the packet, causing massive delays), you're much better off by deactivating the error correction protocol.

Here are some examples of modem init strings:

USRobotics Sportster:	`S0=0&M0&K0`
Global Village PowerPort Mercury:	`S0=0\N0`
Global Village PowerPort Platinum:	`S0=0\N0`
Hayes Accura288 V.FC:	`S0=0&Q0&S46=0`

We'd like to list more modem init strings here for other modems, so if you know what yours would be for your modem and it's not listed here, please send the information to "pgpfone-bugs@mit.edu". The modem init string that you send should tell the modem to turn off error correction and compression and not answer the phone line if it rings. All other settings should remain at their defaults in your modem's memory. Factory settings are assumed to setup hardware handshaking if available, and normal verbose response codes.

Note that PGPfone will not work with Apple's GeoPort Telecom Adapter modems. These modems have an unusual design-- they use the Mac CPU to do the modem processing that all other modems do for you, such as error control and compression. This leaves insufficient CPU time for the intensive computing that PGPfone requires. If you have one of these modems, and you want to run PGPfone, then go out and get yourself a real modem.

Phone Preferences

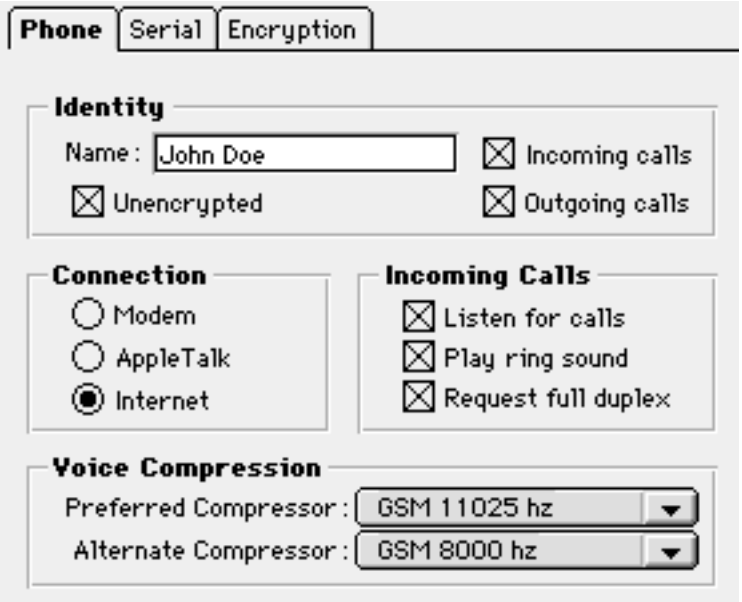

Figure 4: Phone Preferences dialog box

Identity: The user's name (that's you) should be entered in this box. On the Mac, it will default to the machine's File Sharing owner name. The name entered here will appear on the remote user's screen when a call begins, and is sent across the connection after encryption is setup unless you specify that it can be sent unencrypted. If you do not wish to have an identity, several checkboxes have been included to disable this feature in specific circumstances. The "Incoming calls" box enables PGPfone to reveal your identity to the calling party when they call you. The "Outgoing calls" box enables PGPfone to reveal your identity to the called party when you call them. For both incoming and outgoing calls, your identity is revealed only after the encrypted link has been negotiated, so only the other party may see your identifier, not wiretappers. When using PGPfone over the Internet, checking the unencrypted box means your identity is sent before the encrypted link is established, which allows the remote party to know who you are before they accept the call. In that case, the call is still encrypted, but your identity is not.

Listen for calls: Check this box to tell PGPfone that it should open the designated port when PGPfone starts up, and watch the port for incoming calls. The default setting is on, but you can turn it off if desired. If you are using direct modem connections, you might want to turn it off if you plan to run another modem-related application, so that PGPfone doesn't try to have the modem

port open while another application needs the modem port. You should only check this box after you have properly selected the right port for the modem (typically the modem port, unsurprisingly). If you check this box and the modem port is selected wrong, PGPfone will attempt to open the wrong port. Over the Internet and AppleTalk, these issues do not apply.

Play ring sound: Check this box to have PGPfone play a telephone ringing sound over the speaker when an incoming call is detected. This is obviously a good feature to have enabled all the time, and we probably should not have made it a checkbox. The default setting is on.

Request Full Duplex:

Half duplex means you talk like you're using a CB radio, with a push-to-talk mic button. Only one person can speak at a time. Over.

Full duplex means you talk like you normally talk on the telephone, with both parties speaking at the same time. Obviously, this is the preferred way of doing things. However, there are limitations imposed by the type of computer you or the other party is using.

Some speech compressors require more compute power to run in both directions than your computer has available. To use those compressors, you must either have a fast computer, or you must run in half duplex mode. If either party has to run in half duplex, the other party must also run in half duplex, even if he has a fast enough computer to run full duplex. If you have anything less than a very fast computer (like a PowerMac), and you insist on running in full duplex mode, you may have to settle for a less compute-intensive voice compressor that doesn't sound as nice.

Multimedia PCs often have sound cards that are only capable of running in half duplex mode, regardless of how powerful the CPU is. While Macintosh sound hardware is capable of full duplex operation, a Mac talking to a PC often runs in half duplex because almost all PCs run in half duplex. If two Macs are talking to each other, they may run in full duplex mode, but only if both Macs are fast enough to run both their speech compressors and decompressors simultaneously.

If you want to run PGPfone in full duplex mode, it's important that both parties use headphones instead of speakers. Otherwise, there will be audio feedback between the speakers and the microphone, with your voices echoing back and forth many times, with some latency between each echo. This is annoying to most people. It's not enough for you to use headphones -- the other party should too, or you will hear your own voice come back to you from his speakers through his microphone back to your headphones. If you or the other party don't have headphones handy, use half duplex, or both of you can use the microphone "Mute" button on PGPfone's front panel while the other party is speaking.

Only a select few sound cards exist which support full duplex for Windows machines. PGPfone

automatically tests for the existence of full duplex when it starts up. If it is not available, the "Request Full Duplex" checkbox will be dimmed. The cards which support full duplex include the Turtle Beach Tropez, AcerMagic S23, Gravis UltraSound Max, ASB16 Audio System from AdLib, and most recently it became possible to download drivers for the common SoundBlaster 16 for full duplex operation from the Creative Labs web site (www.creaf.com). Not all sound cards are as good as others.

Connection:

This section of the dialog box allows you to set what type of connection you want to run PGPfone over. The choices are Modem-to-modem, Appletalk, and Internet.

Modem-to-modem connections seem to be the most common form of connection. In this mode, your modem is talking to the other party's modem, with only the phone connection between the two modems. Compared to using the Internet mode, this modem-to-modem mode has the advantage of not requiring either party to have an Internet connection. It's a simpler situation.

The Macintosh version supports AppleTalk access which lets you talk to people on your Appletalk network instead of through a modem. Appletalk offers higher bandwidth than a modem, which reduces latency, and a lower transmission error rate. The procedure for placing calls to other AppleTalk nodes is explained later.

Internet calls are also supported. We created PGPfone to allow private conversations between people. The initial release of PGPfone accomplished this by encrypting phone calls between two people via their modems, with a direct connection between the two people's modems, using only the phone system as the intermediary. But popular demand has driven us to add the capability of sending the data stream over the Internet, instead of just the phone system. This feature allows for cheaper long distance conversations, with only the cost of a local phone call to your Internet service provider. I hope this doesn't result in the Internet eventually being glutted by too much voice traffic. Cheaper phone calls is an almost-unintended side effect that will probably not be well received by the long distance phone companies. Our goal was privacy, not cheaper phone calls. Sorry about that, AT&T. But maybe this feature will make PGPfone more popular.

If you enable Internet protocol for PGPfone, and have properly configured access to the Internet on your computer, you may place calls to other users who are also currently connected directly to the Internet and who are currently running PGPfone waiting for a call. The procedure for placing calls to other Internet users is explained later.

If you enable Internet protocol in the PGPfone Preferences, you must be properly connected to the Internet to run PGPfone. When PGPfone starts up, if you have the "Listen for calls" switch enabled, and if you don't already have an active Internet connection running, then PGPfone will remain suspended until your machine establishes a good connection to the Internet.

To connect your computer directly to the Internet, you would typically use a (hopefully fast) modem connection to your Internet service provider, via the PPP or SLIP protocol. For a faster connection, get an ISDN phone line to your Internet service provider. For an even faster connection, you may be connected to the Internet via an Ethernet-ISDN bridge or through Ethernet over a T1 line. If you are not already connected to the Internet the proper way, and you want to use the Internet to make PGPfone calls, you may want to consult an expert on how to get connected directly to the Internet, perhaps by contacting an Internet service provider.

If your computer is connected to the Internet via a modem instead of an ISDN line or an Ethernet bridge, you may sometimes experience increased delays due to your modem's error correction features. We recommend that you turn error correction mode off in your modem when you use PGPfone in the direct modem-to-modem mode, to avoid delays caused by the modems retransmitting bad packets. But when you use PGPfone in Internet mode, and you use your modem to connect your computer to the Internet, your PPP or SLIP software typically asks the modem to place the call to your Internet service provider with your modem's error correction features turned on. This could cause problems if any line noise results in your modem retransmitting the voice packets to the Internet, due to the delays of retransmission.

The Macintosh version of PGPfone requires Apple's Open Transport 1.1 or MacTCP 2.0.6 to run over the Internet. PGPfone is Open Transport native, so using that will give you better performance than MacTCP.

The Windows version requires a Win32 operating system with a Winsock compatible protocol stack such as the Microsoft stack included with Windows95.

Note that sending voice data over the Internet does not have any guarantees of throughput. If the Internet paths between the two parties are glutted with other traffic, your voice packets may not get through in a timely manner, resulting in long latencies and voice dropouts. If you need snappy response and few dropouts, and the Internet connection can't give it to you, you should probably switch PGPfone to a direct modem-to-modem connection.

Encryption Preferences:

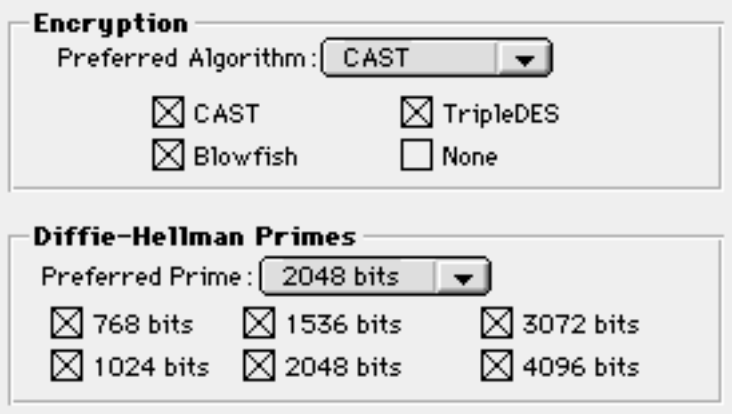

Figure 5: Encryption preferences dialog box

The encryption algorithm is selected as the calling party's preferred algorithm, if that is also supported by the called party. If not, another algorithm that is supported by both parties is chosen. Supported algorithms are determined by check boxes. If both parties select "None" as their preferred encryption algorithm, the call will not be encrypted.

We recommend you leave the preferred encryption algorithm set to CAST, which is probably the strongest of the available choices. CAST and Blowfish are both much faster than triple-DES, and we think that neither of them are weaker than triple-DES. CAST and Blowfish are discussed later in this manual. Triple-DES is so slow that only PowerPC Macs or Pentium PCs can run it with any decent speech compressor like GSM. If you select triple-DES, make sure that both you and the other party are using a fast computer like a PowerMac or Pentium. If you have a PowerMac, and the other party has a slower 680x0 Mac, he may not be able to triple-DES decrypt your voice stream to him and still have the CPU cycles to spare to run the better speech compressors at his end.

We also recommend you leave your preferred Diffie-Hellman prime to the default setting, 2048 bits. If one party has a larger preferred prime than the other party, the larger of the two is selected, assuming both parties support that size. Using the largest sizes can result in long periods of calculation at the beginning of a call. There may be some security advantages in selecting a 3072 bit prime, at a cost of waiting longer for the initial negotiation. Larger primes may yield diminishing returns.

Answering and Placing Calls with PGPfone

Answering a call

When PGPfone is waiting for a call, and the phone rings, the "Dial" button changes its name to "Answer". If you push that button, PGPfone will answer the call, and establish a connection,

assuming that the caller is also running PGPfone at the other end. It negotiates with the other party's PGPfone software to agree on an encryption algorithm, a key, full or half duplex, and a voice compression algorithm. Then the conversation can begin between the humans.

If the call is full duplex, you can just talk and listen at the same time. If it's half duplex, you will see a button that either says "Click to Talk", or "Talk Now". Do what comes naturally with that button, clicking it once to toggle between talking and listening. Begin your conversation.

When you're done, press the "Hangup" button to end the call.

Of course, if the other caller is not running PGPfone, but was just a human calling you to talk without a computer, they will hear a rude modem squeal when PGPfone answers. This is a fun way to discourage those pesky telemarketers.

Placing a call by modem

To place a call over a direct modem connection, type in the phone number in the number field on the main PGPfone window and hit the "Dial" button. PGPfone will use the modem to place the call, hopefully to another modem connected to a computer running PGPfone at the other end. The phone will ring at the other end, and the other party will have to press their own "Answer" button to pick up the phone. The two parties' modems will establish a modem connection, then the two PGPfone programs will begin negotiating an encryption algorithm, a key, full or half duplex, and a compression algorithm. Then the conversation can start.

If the call is full duplex, you can just talk and listen at the same time. If it's half duplex, you will see a button that either says "Click to Talk", or "Talk Now". Do what comes naturally with that button, clicking it once to toggle between talking and listening. Begin your conversation.

When you're done, press the "Hangup" button to end the call.

Placing a call on an AppleTalk network

When you have Appletalk connectivity enabled in the PGPfone Preferences, you don't type in a phone number in the main display window. Instead, the "Dial" button in the main display window changes to a "Connect..." button. When you press the "Connect..." button, PGPfone presents you with a separate dialog box with a scrolling list of Appletalk nodes that currently have PGPfone running, waiting for a call. When you select one of those Appletalk nodes, PGPfone places a call to that node.

Placing a call on the Internet

You can make calls over the Internet. If you enable Internet protocol in the PGPfone Preferences, and you are properly connected directly to the Internet, you may place calls to other users who are also currently connected directly to the Internet and who are currently running PGPfone and are waiting for a call. To make such a call, you must know the IP address or the domain name of the person you are calling. When you type in the IP address or the domain name in the main display window and press the "Connect" button, PGPfone will place a call to that node. If that node is currently connected to another Internet user, you will hear a busy signal through your computer's speaker. If the remote user is waiting for a call, you will hear a ringing sound as if you were calling someone over a normal phone line.

Switching to PGPfone during a normal voice call

It's possible to start a normal old-fashioned telephone conversation without PGPfone, without your computer, and then decide that the topics you are discussing would better be discussed over a secure channel. Assuming you are using the same phone line that has your computer and modem and PGPfone connected to it, you can "go secure" by letting PGPfone take over midway through the phone call. For this to work, both parties must have PGPfone up and running. The method you use to go secure is different for the two parties. We must digress a bit to explain why.

To make the modems start talking to each other, you have to bear in mind that modems whistle at each other with different tones or frequencies; one modem emits an originator tone while the other emits an answer tone. Normally, the modem which detected the ring and answered the phone line implicitly knows it is the answerer, so it emits the answer tone, while the modem that dialed the call implicitly knows it is the originator, and thus should listen for the answer tone from the other modem and then should respond with the originator tone.

But when you want to have PGPfone "go secure" and take over midway through a normal voice phone call, the two modems do not implicitly know which is the originator or the answerer, because the humans manually dialed the call and manually answered the ringing phone without either of the modem's participation. This means that the two human parties must agree between themselves which of them will act as the originator, and which will act as the answerer, so that they can tell their modems which role to assume. The originator's PGPfone will tell his modem to go into originate mode, while the answerer's PGPfone tells his modem to go into answer mode. They must not both tell their respective modems to be originators, nor can they both tell their respective modems to be answerers. One must be the originator, the other must be the answerer. The human users cause this to happen by each pushing different buttons on their respective PGPfone front panels, at roughly the same time. Since the two parties were already talking directly to each other though the phone line, they can first verbally agree which will be the originator and which will be the answerer, and then both invoke the appropriate choice at about the

same time. The simplest convention is to always have the party who initiated the call to act as the originator.

The designated originator should click the "Dial" button on PGPfone's front panel. Clicking that button will cause his modem to go into originate mode and try to make contact with the other modem, which hopefully is in answer mode. The user should then hang up his regular phone and let his modem take over.

At about the same time, the designated answerer should select "Answer" from PGPfone's pull-down menus. This will cause his modem to go into answer mode and try to make contact with the other modem, which hopefully is in originate mode. The user should then hang up his regular phone and let his modem take over.

Then the two squealing modems will establish a normal modem link, just as they would if the modems dialed and answered the phones from the beginning of the call. Then the two PGPfones will negotiate the rest of the call parameters, and a secure conversation can then proceed. Whew.

If all this seems excessively complicated, bear in mind that a secure phone built entirely from dedicated hardware has a modem built into the phone, and those phones would always implicitly know who originated the call and who answered it, and thus they can control their built-in modems accordingly. But PGPfone is a pure software product that runs on an installed base of personal computers and modems. This means that if the call was manually started without PGPfone's involvement from the beginning of the call, we have to go through these extra complexities to control the modems. Actually, even though the explanation seems complex, the actual operation is not that bad. The only minor "complexity" is that each of the two parties invoke a different function to make PGPfone take over the call.

Silence Detection

PGPfone can be configured to send sound over the channel when you talk, and not send sound when you stop talking. A triangle control below the sound input meter allows you to set the threshold of sound volume above which sound should be sent. This allows channel bandwidth to be saved by only sending and compressing sound which is above a certain volume. This is a nice feature that should cut down on how many packets must be sent over a shared network, such as the Internet. A green light next to the input meter illuminates whenever sound is being sent. To move the threshold, click the spot on the bottom edge of the input meter that you'd like the threshold set at. If you move it all the way down to the left, silence detection is turned off and sound is sent continuously. If you move it all the way to the right, no sound can be sent, which is equivalent to using a mute button.

In the current release, this silence detection feature is implemented on the Mac, but not on

Windows.

Biometric signature checking

PGPfone will display some special words to authenticate the call, and it is possible to use these words as a means of ensuring that nobody is tapping in. You should read these authentication words aloud to the other party, and they should read them back to you, to determine that both of you have the same authentication sequence displayed on your screens. If the words match at both ends, and you clearly recognize that it's your friend's voice reading the authentication words, then you have good reason to believe that the call is secure and no one is tapping in. See figure 6 below.

Figure 6: Authentication dialog

If the words don't match, or it doesn't sound like your friend's voice reading the words, then something is terribly wrong. Either a catastrophic bug has appeared in PGPfone, or someone has mounted a sophisticated computer-assisted wiretap to intercept this call. I presume this is fairly unlikely. (During the beta test phase of PGPfone, a bug seems a far more likely explanation.)

It may seem odd to many people that reading some magic words aloud can somehow help you detect a wiretap. Why not swing a dead cat under a full moon, or use a Ouija board? Well, we use this method to authenticate the key exchange to prevent the "man-in-the-middle" attack, which is explained in the appendix.

The list of special authentication words that PGPfone uses is carefully selected to be phonetically distinct and easy to understand without phonetic ambiguity. The word list serves a similar purpose as the military alphabet, which allows pilots to convey information distinctly over a noisy radio channel.

If you'd like to know how this biometric signature technique really helps prevent wiretapping, you can find out the details in the appendix.

A Word About Modem Quality

The biggest problems we have observed during PGPfone testing are related to modems. Most people judge modems by how they perform during ordinary data transfers such as file downloads. All modems that are capable of supporting PGPfone have error correction features such as V.42bis, which involves the modem detecting errors and retransmitting the data. This method works great for downloading files, but it isn't so great for real-time voice packet streams, because of the increased latency. You might feel you have a high quality modem because it has always worked reliably for downloading files. Even if your communication software uses a reliable file transfer protocol such as Zmodem, chances are that Zmodem's error detection features will never even see any errors, because the modem corrects them before Zmodem can see them. This error correction feature that modems have can mask other transmission problems of the modem. In a really good modem, the errors should not even occur at all, and thus should not require any retransmissions unless the phone line is just too noisy. But you won't see these kinds of problems with your modem until you run PGPfone, which generally has to run with the modem's error correction features turned off. You might think these problems are caused by PGPfone, but the problems were there all along, and you never saw them before because the modem's error correction mode hid them from you.

Another issue that depends on modem quality is what speed it can run at on a particular connection. If the line quality is not good enough to sustain the highest modem speed, the modem drops down to a lower speed. This is OK if all you want to do is download a file, but PGPfone needs all the modem bandwidth it can get to run the compressed voice stream through. PGPfone really needs a *real-time* data stream to work. If your modem falls back to a lower speed, the speech quality will suffer. If the modem speed falls low enough, none of the speech coders will work and you just can't carry on a conversation at all. A marginal phone line will make a cheaper modem drop to a lower modem speed more readily than a really high quality modem.

There are other secure telephones that use dedicated hardware, such as the STU-III from Motorola or AT&T, or the AT&T Surity 3600 (the "Clipper" phone). These phones have the modems built in, so the modem was engineered to work together with the rest of the phone (of course, you might also consider that these products cost anywhere from $1200 to $3000, plus they have back doors to help the government wiretap them). PGPfone is a piece of software that runs on the installed base of computers, and so anyone can buy a dirt-cheap modem out of a mail-order catalog and hang it out on the end of a modem cable and then they expect PGPfone to work with it.

If you are serious about having good reliable secure voice telephony, you must buy a good modem to run PGPfone through. A communication software package that downloads your email can handle a cheap modem, but PGPfone can't. PGPfone requires the data to be delivered in real time, with very few errors, and that means you just can't skimp on the modem.

Macintosh Sound Input/Output

The sound input features of Macintosh computers have varied widely through the lifetime of the Macintosh. Beginning around the release of the PowerBook 180 series, most Macs began to support full duplex sound.

The microphone jack on the back of your Macintosh is one of two types. If you have a relatively new Macintosh that supports Apple's PlainTalk microphone, it is the newer line-level version of the input jack; otherwise, you have the mic-level input jack. Getting a normal microphone to work with the line-level jack can be a nightmare. In general, the simplest way to use PGPfone is to use whichever Apple supplied microphone you have and plug in a pair of headphones. Using PGPfone without headphones is generally a bad idea especially in full duplex because an audio feedback echo can occur. The sound quality is also usually much better with good headphones as opposed to the built in speaker.

You might think that any simple computer telephony microphone headset such as the inexpensive Labtec C-15 would work with your line-level Macintosh. Sorry. You will either need to use PGPfone's sound input amplifier on the front panel or obtain a device to boost the faint mic-level signal from a normal microphone up to the line-level signal that your computer expects. This is a complicated issue, and you may wish to contact an audio specialty store if you choose to get a better microphone. There are several options. A few products on the market designed for videoconferencing such as the VideoLabs FlexCam come with built-in microphones. The FlexCam microphone is even stereo (of course, the current version of PGPfone wouldn't actually take advantage of that). Be careful not to use products such as the Connectix QuickCam which have extremely low quality microphones. If you have your own microphone, you may need to obtain a microphone preamplifier. These can range in price from (US) $50 to $500.

One particularly good setup that Will Price put together on a line-level Macintosh involved a DBX 760X Microphone Preamplifier with a Roland RHS-300 stereo microphone headset. Those two items together will run you about $400. This is undoubtedly more than most PGPfone users will want to spend, but to hear a secure phone call with that kind of audio fidelity over two PowerMacs running with GSM 11025 is quite impressive. This is not to be considered an endorsement. It is just to note that there is a wide range of equipment available which can turn your Macintosh into a very high sound-quality secure telephone.

If you are a PowerBook 500 series user, you'll get better results by spinning down your hard drive during a call if you are using the built-in microphone. The microphone on these computer is right next to the hard drive, so spin noise from the drive can produce loud background noise. Put your ear next to the keyboard where the microphone is to hear just how loud the hard drive can be and then imagine how the microphone feels sitting there all the time. You can easily spin down your hard disk during a call using the control strip at the bottom of the screen.

Windows Sound Input/Output

Windows 95 has a confusing array of system level sound control panels for volume and microphone input. Before running PGPfone under a Windows OS, it is recommended that you confirm your microphone is properly configured and has enough gain to record the sounds you make. To do this, use the *Sound Recorder* application that came with the OS which allows you to record sounds. Once you're sure that your microphone is recording at an acceptable level, it should work with PGPfone. Correctly configuring your microphone can vary from system to system and sometimes it varies with different types of sound cards, but most of the settings you will need will be accessed by double clicking the volume control from the taskbar:

Figure 7: Win95 sound control

If your sound card allows you to turn on automatic gain control, you should activate that. Make sure that you have a mic-level microphone hooked up to the microphone jack on your sound card rather than the line jack. Hooking it up to the wrong jack is a common source of problems.

Tips for regularly using your PGPfone

I have collected a few business cards from some government employees, particularly NSA employees, that list a separate phone number for their secure phone lines. I think this is kind of cool. PGPfone users should be able to do the same.

If you plan to make your PGPfone available to for incoming calls on a continuous basis, and you plan to use it in the modem-to-modem mode instead of the Internet mode, you might want to get a dedicated phone line for it. Especially if you want to list it on your business card.

If you want to list a separate phone number for your PGPfone, but you don't want to pay for a whole separate phone line just for that, there is a cheaper alternative. Get a custom ringing phone number from your phone company, which costs a lot less than a separate phone line. Custom ringing is available in most areas of the USA. It's sort of like having two separate phone lines. Two phone numbers both share the same phone line, but each rings in a different ringing pattern. The main number rings in the usual way, but the custom ringing number rings in a distinctive ringing pattern of two short rings, a pause, two short rings again, a pause, et cetera. This lets you know which phone number is being called. Some people use this feature to share a single phone line between voice and fax. You can buy a variety of commercial hardware products that attach to your phone line and route your incoming calls to two different devices, depending on the ringing

pattern. You can set it up to route the normal ring to your normal telephone, and the custom ring to your PGPfone-equipped computer. I bought one of these routing devices through one of those airline in-flight mail-order catalogs.

Vulnerabilities

No data security system is impenetrable. PGPfone 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.

Man-in-the-middle attack

PGPfone uses the Diffie-Hellman (DH) exponential key exchange to agree on a shared key between the two parties to allow an encrypted conversation to take place. The biggest vulnerability that the DH algorithm has is the man-in-the-middle attack.

When two users, Alice and Bob, use a DH exchange to agree on a common key between them to set up an encrypted channel, no one can listen in after the encrypted channel is set up.

But suppose an eavesdropper, Eve, interposes herself on the phone line between Alice and Bob. Eve does not merely listen passively, but actually inserts her own traffic on the line to both Alice and Bob. In fact, she mascarades as Alice to Bob, and she mascarades as Bob to Alice. I'm not talking about Eve changing her voice -- I mean she does this digital mascarade just during the DH key negotiation when the call is set up initially.

Eve uses DH to negotiate a shared key to set up an encrypted channel with Bob. Bob mistakenly believes he has negotiated a key with Alice. At the same time, Eve uses DH to negotiate a shared key to set up an encrypted channel with Alice. Alice mistakenly believes she has negotiated a key with Bob. The Alice-Eve key is different from the Bob-Eve key. They are two different encrypted channels, with two different keys. After the encrypted channels are set up, Eve just sits in the middle, passively listening to the decrypted voice streams from both Alice and Bob, passing them through to each other. She can accomplish this by merely operating two PGPfones back to back, one PGPfone connected to Alice, the other to Bob. The decrypted voice streams are just passed through in both directions, so that Alice and Bob can hear each other's voices. No cryptanalysis is

involved, just deception.

There is an effective countermeasure to this man-in-the-middle attack. In a normal unmolested DH key negotiation, Alice and Bob end up sharing a common key between them that the DH algorithm computes for them, and they use that shared key to set up the encrypted channel. If Eve does the man-in-the-middle attack, then Alice and Bob do not truly have a shared key. They have two different keys, each of them encrypting a separate channel to Eve. A simple way to detect this dastardly deception is for Alice and Bob to verbally tell each other what key they are using for their encrypted channel. The keys had better agree. If they do not, then there is a man-in-the-middle attack in progress.

So Alice and Bob could authenticate the key exchange by reading some binary key material to each other as hexadecimal bytes. This is fine for computer geeks, but awkward for nontechnical humans. And prone to errors. So PGPfone encodes each byte as one of 256 phonetically distinct English words, kind of like the military alphabet that pilots use to read letters over a noisy radio channel. You know, that "tango delta foxtrot" stuff. Only we use a bigger list of words, 256 instead of 26. We call this a "biometric signature", because we are using the human vocal tract as a way of authenticating or "signing" the information. It helps if Alice and Bob recognize each other's voices. Actually, even if you've never spoken to the other party before, this biometric signature scheme can still work, as long as the voice pronouncing the authentication words clearly matches the voice in the rest of the conversation.

So when you make a call with PGPfone, PGPfone will display a sequence of words that you should read aloud to the other party to make sure they are the same words at both ends of the call. If they do not match, an incredibly sophisticated (but apparently not quite sophisticated enough) wiretap may be in progress (either that, or PGPfone has a serious bug).

There is still one difficult ploy that Eve can do to pull off the attack anyway. She can imitate Bob's voice to Alice, and Alice's voice to Bob, reading a different authentication word sequence to each of them. I call this the "Rich Little" attack (named after a voice impersonator who did a really great Dick Nixon). This is a daring attack -- meaning there is a high risk of the attack being detected.

Viruses and Trojan Horses

Another attack could involve a specially-tailored hostile computer virus or worm that might infect PGPfone or your operating system. This hypothetical virus could be designed to capture your session key, and covertly write the captured information to a file or send it through a network to the virus's owner. Or it might alter PGPfone's behavior so that session keys are written out to the channel encrypted with a key held by the attacker, in a manner similar to the Clipper chip's Law Enforcement Access Field. 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. PGPfone 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 PGPfone that behaves like PGPfone in most respects, but doesn't work the way it's supposed to. For example, it might transmit session keys on the channel, in a manner similar to how the Clipper chip does it. This "Trojan horse" version of PGPfone is not hard for an attacker to create, because PGPfone source code is widely available, so anyone could modify the source code and produce a lobotomized zombie imitation PGPfone that looks real but does the bidding of its diabolical master. This Trojan horse version of PGPfone could then be widely circulated, claiming to be from me. How insidious.

You should make an effort to get your copy of PGPfone from a reliable source, such as MIT. Or perhaps from more than one independent source, and compare them with a file comparison utility.

There are other ways to check PGPfone for tampering, using digital signatures. You can use PGP for this purpose. If someone you trust signs the executable version of PGPfone, vouching for the fact that it has not been infected or tampered with, you can be reasonably sure that you have a good copy. But this will not help at all if your operating system is infected, nor will it detect if your executable copy of PGP 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 PGPfone executable.

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

The danger of hostile software infecting your computer is probably the main reason why the NSA recommends dedicated encryption hardware to protect classified data. The government's STU-III (Secure Telephone Unit) is a hardware device, not a general-purpose programmable computer. No hostile software can infect it. But this hardware approach, although more secure, is also more expensive. PGPfone is intended to reach the masses, with the ability to replicate and spread like

dandelion seeds in the wind. The only approach that can accomplish this is pure software running on general purpose computers.

Physical Security Breach

A physical security breach may allow someone to physically access your premises and plant a bug. He might also use the opportunity to acquire other information, such as documents or files stored in plaintext form. 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 PGPfone.

Listening devices and other remote sensing

Obviously PGPfone cannot protect you from someone planting a microphone in the room where you are speaking on an encrypted phone call. Acoustic information may leak out of your room by other means, too. For example, from hundreds of feet away, someone may bounce a laser beam off your window, and watch the beam's reflection wiggle as your voice makes the window vibrate. They can use the reflected beam's wiggle to reconstruct the sound. If they use an infrared laser, you won't be able to see the beam on your window.

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

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. In some cases it may even detect data as it moves through your computer's bus or I/O cables. This could compromise some of your

passwords, messages, or multimedia streams such as digitized voice, 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?

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 Diffie-Hellman session key by using some new secret breakthrough in computing discrete logarithms. Perhaps so, but it is noteworthy that the US government trusts the Diffie-Hellman key exchange algorithm enough to use it in their own STU-III Secure Telephone Unit, used for voice communications of classified information. And civilian academia has been intensively attacking Diffie-Hellman without success since the late 1970s.

Perhaps the government has some classified methods of cracking one or more of the symmetrical single-key bulk encryption algorithms used in PGPfone. This is every cryptographer's worst nightmare. There can be no absolute security guarantees in practical cryptographic implementations.

PGPfone uses some carefully selected strong block ciphers in its design, and they are briefly discussed below in the section on block ciphers.

In summary, without good cryptographic protection of your voice or data communications, it may have been practically effortless and perhaps even routine for an opponent to intercept your electronic traffic. If you use PGPfone 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 PGPfone. PGPfone gives you pretty good privacy.

Block ciphers used by PGPfone

The voice stream in PGPfone is encrypted with one of three block ciphers: CAST, Triple DES, or Blowfish. I thought it would be a good idea to talk a bit about why I selected these particular ciphers to be used in PGPfone.

Selecting a cipher for use in PGP or PGPfone is a weighty decision for me, because I know that these products are sometimes used in situations where formidable resources may be applied to breaking them. In the case of PGP, which encrypts E-mail, choosing a strong cipher is especially important, because once the cipher has been designed in, it becomes difficult to remove it because of the "legacy burden" of having to decrypt archived messages years later. With PGPfone there is less of a legacy problem, because conversations are not archived and encryption algorithms are negotiated between the two parties on the fly with every phone call. This means that new encryption algorithms may be added with any new release, and any encryption algorithms that have weaknesses discovered may be easily removed from later releases. Nonetheless, I am very careful about selecting block ciphers for PGPfone.

I included the CAST encryption algorithm in PGPfone because it shows promise as a good block cipher, it's 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 committment 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 semiweak 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 the DES. While CAST is too new to have developed a 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 PGP.

I also included the Blowfish encryption algorithm for the first release of PGPfone because it seems to be well-designed, it's fast, and it is not patented. Blowfish was designed in 1995 by Bruce Schneier, who also wrote the book *Applied Cryptography*. Bruce took a nice safe conservative approach to designing Blowfish, but it does not have quite as formal a design approach as CAST. I asked Eli Biham, who cracked the DES with differential cryptanalysis, what he thought about Blowfish, and he did not see any vulnerabilities to a differential cryptanalytic attack. Blowfish appears to be designed to withstand both differential and linear cryptanalysis. Blowfish does not yet have the long track record that is usually required to earn the trust of cryptographers for a new encryption algorithm, but no problems have been uncovered so far.

As a hedge, I also included Triple-DES in PGPfone's repertoire of available block ciphers. Triple-DES is very strong, and has been well-studied for many years, so it might be a safer bet. But

Triple-DES is much slower than CAST or Blowfish, leaving less processing power available for speech compression. Triple-DES appears to have about 112 bits of key strength.

I had considered adding the IDEA cipher to PGPfone, but ruled it out because of Ascom-Systec's patent licensing terms for using the IDEA algorithm. That's too bad, because IDEA is a very good cipher that has withstood repeated attacks in the academic literature. Perhaps Ascom's licensing policies will change in the future.

Legal Issues

Copyrights, Trademarks, Warranties

PGP®, Pretty Good Privacy™, PGPfone™, Pretty Good Privacy Phone™, Phil's Pretty Good Software™, and the Pretty Good® label for computer software and hardware products are all trademarks of Pretty Good Privacy Inc.

The PGPfone software for Macintosh, Windows, and other environments are all copyright ©1995-1996 Pretty Good Privacy Inc. All rights reserved. The PGPfone Owner's Manual is also copyright ©1995 Pretty Good Privacy Inc. All rights reserved. These rights include but are not limited to any foreign language translations of the manual or the software, and all derivative works of both.

The abovementioned trademarks and copyrights were formerly held by Philip Zimmermann.

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

Both Pretty Good Privacy Inc and Philip Zimmermann assume no liability for damages resulting from the use of this software, even if the damage results from defects in this software, and they make no representations concerning the merchantability of this software or its suitability for any specific purpose. It is provided "as is" without expressed or implied warranty of any kind.

Copyrights for other software components

PGPfone includes GSM speech compression routines, which are derived from code that is copyright 1992 by Jutta Degener and Carsten Bormann, Technische Universitaet Berlin.

The TripleDES code in PGPfone was provided by Richard Outerbridge, and is copyrighted by Richard Outerbridge.

PGPfone includes some ADPCM voice compression subroutines with permission, under the following copyright notice.

> ADPCM software is Copyright 1992 by Stichting Mathematisch Centrum, Amsterdam, The Netherlands. All Rights Reserved
>
> Permission to use, copy, modify, and distribute this software and its documentation for any

purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the names of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Patents

PGPfone uses the Diffie-Hellman exponential key exchange algorithm for key management, which is claimed by patents in the US and Canada. Cylink Corporation holds exclusive sublicensing rights to these patents. Pretty Good Privacy Inc has obtained the necessary patent licenses from Cylink which allows Pretty Good Privacy Inc to make, use, and sell PGPfone and other products that use the Diffie-Hellman algorithm.

The overall patent situation in cryptography is complicated, so let's explore that a bit. We'll talk about the Diffie-Hellman patent, as well as some other patents that are not used by PGPfone at present, but are nonetheless noteworthy.

Most of the useful public key algorithms are claimed by patents. Until recently, these patents were controlled by Public Key Partners (PKP), until PKP broke up in September 1995. Jim Bidzos, the president of RSA Data Security Inc (RSADSI), was also the president of PKP before it broke up. There are five patents formerly held by PKP:

Patent #	Title	Granted to	Expires
4,200,770	Cryptographic Apparatus and Method ("Diffie-Hellman")	Stanford	March 29, 1997
4,218,582	Public Key Cryptographic Apparatus ("Hellman-Merkle")	Stanford	August 19, 1997
4,405,829	Cryptographic Communication System and Method ("RSA")	MIT.	September 20, 2000
4,424,414	Exponential Cryptographic Apparatus and Method ("Hellman-Pohlig")	Stanford	March 3, 2001
4,995,082	Method for Identifying Subscribers... ("Schnorr")	C. Schnorr	February 19, 2008

Of these five patents, all but the Schnorr patent were developed with at least partial US government funding. This gives the government the right to practice those patents. The RSA algorithm was developed at MIT by Ronald Rivest, Adi Shamir, and Len Adleman. The DH algorithm was developed at Stanford University by Whitfield Diffie and Martin Hellman. The Hellman-Merkle algorithm was also developed at Stanford. The Hellman-Pohlig algorithm was also patented by Stanford, but is not a public-key algorithm and thus is not relevant to this discussion.

Stanford University granted an exclusive license to its patents to Cylink Corporation. MIT granted its RSA patent to RSA Data Security, Inc (RSADSI), a company started for the purpose of exploiting the RSA patent. Cylink and RSADSI pooled their patents to form a new partnership called Public Key Partners (PKP), headed by Jim Bidzos, also the president of RSADSI.

There are several problems with the patent situation, from the point of view of the patent holders. This requires some explanation regarding PKP, Cylink, and RSADSI, and their overall patent situation. It's rather complicated, so I'll try to explain it to the best of my knowledge.

In September 1995, PKP dissolved, as a result of a lawsuit filed by Cylink against RSADSI, one of the partners in PKP. Cylink's suit claims, among other things, that the RSA patent is invalid. When PKP broke up, and the patents reverted to their respective original patent holders. RSADSI got back the RSA patent. Cylink got back the Stanford patents. The Schnorr patent reverted to Schnorr, but was licensed again to RSADSI. Since the breakup, Cylink has filed other suits against RSADSI. Interestingly enough, Cylink's in-house lawyer, Bob Fougner, used to be the in-house lawyer for PKP before it broke up.

The Hellman-Merkle patent makes broad claims on all forms of public-key cryptography. It is

based on the famous Knapsack algorithm, which was broken by Adi Shamir in the early 1980s. Since the breakup of PKP, Cylink has asserted that RSA licensees must also license the Hellman-Merkle patent from Cylink, because it claims all forms of public key cryptography, including RSA.

The RSA patent remains in effect until the year 2000. Nonetheless, one of the Cylink suits against RSADSI asserts that the RSA patent is invalid.

The Schnorr patent is relatively new, and Bidzos claims that the Schnorr patent is infringed by the NIST Digital Signature Standard (DSS). I don't know of anyone outside of RSADSI that believes the Schnorr patent is infringed by the DSS. Since no government funding was involved when Schnorr developed his patent, the government has no license to use the Schnorr patent. But NIST is claiming that their own DSS patent infringes no other patents, including the Schnorr patent, and will indemnify any government contractors for supplying DSS products to the government. The government has granted a royalty-free license for anyone to use their DSS patent. Further, since the breakup of PKP, Cylink has also asserted that the Schnorr patent is not infringed by the DSS. In fact, Cylink is indemnifying its own customers against any claims of infringement of the Schnorr patent that arise from using the DSS. Cylink's lawyer, Bob Fougner, who decided to offer such indemnification, is the same lawyer who worked on Bidzos's side when PKP held the Schnorr patent. Recently Cylink filed some legal action against RSADSI to stop any claims that RSADSI may attempt against users of the DSS.

Some people have asked me why I did not use the IDEA cipher in PGPfone, since I used it in PGP. The reason is that a Swiss company, Ascom-Systec, holds a patent on the IDEA algorithm. After PGP made the IDEA cipher famous, Ascom raised their prices for licensing their IDEA patent to others. They started charging about US$100 for PGP end-users in Europe just to use the IDEA cipher in the freeware version of PGP. I think that it is counterproductive to use an expensive patented algorithm in a standard if alternatives exist, so I left IDEA out of PGPfone. Patents and standards don't mix.

Licensing

PGPfone is copyrighted freeware. Published as a community service (with only minor restrictions, described below). Giving PGPfone away for free will encourage far more people to use it, which will have a greater social impact. I want this software to spread as widely as possible to the American public. Feel free to disseminate the complete unmodified PGPfone release package as widely as possible, but be careful not to do so in a manner that might violate US cryptographic export controls if you live in the USA. Give it to all your friends. I impose no restrictions of my own on the precise manner you use to disseminate it, but the Feds might, so if you want to make sure you don't get in trouble with the Feds, you should seek legal advice before doing something that might result in a violation of US export laws. But don't ask me for legal advice on this subject.

You may also disseminate the complete unmodified source code release package, again with the caution not to violate US cryptographic export controls. PGPfone's source code is published only to assist public scrutiny of PGPfone to show that it has no hidden weaknesses or back doors, and to help people to find bugs and report them. The source code is not published to give it away for free to others who are developing products.

I place no restraints on your modifying the source code for your own personal experimental use. However, do not distribute a modified version of PGPfone without first getting permission from me. Please respect this restriction. PGPfone's reputation for cryptographic integrity depends on maintaining strict quality control on PGPfone's cryptographic algorithms and protocols. Beyond that, ad hoc "improvements" to PGPfone can affect interoperability, which creates user confusion and compatibility problems that could damage PGPfone's (and my own) reputation and undermine the good will earned by the PGPfone trademark. The source code to PGPfone was not published to help spawn these mutant strains that some users may unilaterally turn loose upon the world. If you feel you must modify your copy of PGPfone, please make sure your modified version cannot escape into the environment. This most especially applies to changes that might affect the security of PGPfone.

This doesn't mean I don't want people to help me change PGP. On the contrary. PGPfone needs many improvements, and I welcome any changes suggested by the user community that seem like good ideas. If you have ideas for changing PGPfone, please let me know. Perhaps we'll decide that your changes are appropriate for incorporating into the standard PGPfone release. I especially need help with better speech compression algorithms and methods for improving the management of voice packets to ensure smooth speech transmission and reduce latency.

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

Restrictions on Commercial Use of PGPfone

There will be two versions of PGPfone available: a freeware version for noncommercial use, and a commercial version that you should buy if you want to use it for commercial purposes. This beta test version of PGPfone is only for noncommercial use. When we do the full release, a commercial version will be made available.

In the meantime, this version of PGPfone may be used temporarily for evaluation purposes in commercial environments. When commercial users learn of a commercial version becoming available to them, they should switch to the commercially licensed version and discontinue using

this version in commercial environments.

If you use any version of PGPfone in such a manner that you must pay patent royalties or any other software licensing fees to any patent holders for any algorithms used by PGPfone, cryptographic or otherwise, then we must agree on a way for me to also be paid in some manner.

Other Licensing Restrictions

Under no circumstances may PGPfone be distributed without the PGPfone documentation, including this PGPfone Owner's Manual. You must also keep the copyright, patent, and trademark notices on PGPfone and its documentation.

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

Distribution

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

The primary release site for PGPfone is the Massachusetts Institute of Technology, via their World Wide Web site, at http://web.mit.edu/pgp, or http://web.mit.edu/network/pgpfone. You may obtain free copies or updates to PGPfone from this site, or any other Internet FTP site or World Wide Web site or BBS that PGPfone has spread to. Don't ask me for a copy directly from me, especially if you live outside the US or Canada. I recommend that you not use any modified version of PGPfone that comes from any other source, other than MIT or me, unless it is accompanied by a PGP digitally-signed endorsement from Jeffrey I. Schiller (of MIT) or myself. You can get the official release software from many other distribution sites "downstream" from MIT. Hopefully, all these other sites are adhering to US export controls.

The PGPfone version 1.0 executable object release package for MacOS or Windows95 or NT contains the PGPfone executable software, documentation, some PGP public keys for checking the integrity of the PGPfone release package, and signatures for the software and this manual, all in one Macintosh self-extracting archive called PGPfone10.sea, or a PKZIP compressed file called pgpf10.zip. The PGPfone source release package contains all the C source files in one StuffIt compressed file called pgpf10s.sit with both Macintosh and Windows source, or as a PKZIP compressed file called pgpf10s.zip with just the Windows source. The filename for the release package is derived from the version number of the release.

Export Controls

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

The US government is using export restrictions as a means of suppressing both domestic and foreign availability of cryptographic technology. In particular, it is trying to suppress the emergence of an international standard for cryptographic protocols, until it can establish a key escrowed standard (such as the Clipper chip) as the dominant standard.

Any export restrictions on PGPfone are imposed by the US government. This does not imply that I or MIT agree with these restrictions. We just comply with them. We do not impose additional licensing restrictions of our own on the use of PGPfone outside of the US, other than those restrictions that already apply inside the US. PGPfone may be subject to export controls. Anyone

wishing to export it should first consult the State Department's Office of Defense Trade Controls.

I will not export this software out of the US or Canada in cases when it is illegal to do so under US controls, and I urge other people not to export it on their own. If you live outside the US or Canada, I urge you not to violate US export laws by getting any version of PGPfone in a way that violates those laws.

At the time of this writing, we do not have an official comment from the government about the exportability of PGPfone. But we may be able to infer their position by looking at the language they supplied to ViaCrypt, a company that sells PGP commercially in the US and Canada, to be inserted into the documentation for PGP: "PGP is export restricted by the Office of Export Administration, United States Department of Commerce and the Offices of Defense Trade Controls and Munitions Control, United States Department of State. PGP cannot be exported or reexported, directly or indirectly, (a) without all export or reexport licenses and governmental approvals required by any applicable laws, or (b) in violation of any prohibition against the export or reexport of any part of PGP." The government may take the position that PGPfone is also subject to those controls.

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

If, despite the best efforts of MIT and myself to prevent export, someone still exports PGPfone, and it spreads overseas, I do not want to get calls from people outside the US who might ask me if it is legal to use PGPfone in their own country. That question is not up to me, and it puts me in an uncomfortable position. I've had enough legal problems of my own with export control issues, without getting involved in giving you legal advice over my telephone. It might even put me at some legal risk to simply answer a question like that for a foreigner. If this question concerns you, ask someone else, like a lawyer.

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

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

Philip Zimmermann's Legal Situation

On January 11th 1996 my lead defense lawyer, Phil Dubois, received a letter from the Assistant US Attorney in the Northern District of California, William P. Keane. Here is the text of the letter:

> "The U.S. Attorney's Office for the Northern District of California has decided that your client, Philip Zimmermann, will not be prosecuted in connection with the posting to USENET in June 1991 of the encryption program Pretty Good Privacy. The investigation is closed."

The US Attorney also released this to the press:

> "Michael J. Yamaguchi, United States Attorney for the Northern District of California, announced today that his office has declined prosecution of any individuals in connection with the posting to USENET in June 1991 of the encryption program known as Pretty Good Privacy. The investigation has been closed. No further comment will be made by the U.S. Attorney's Office on the reasons for declination."

This brought to an end a three-year criminal investigation for an alleged violation of the Arms Export Control Act. Encryption software such as PGP is regarded as a munition by the government, and is subject to export controls. The federal mandatory sentencing guidelines for this offense specify 41 to 51 months in a federal prison. US Customs had taken the position that electronic domestic publication of encryption software such as PGP is the same as exporting it.

I'd like to thank all the people who helped us in this case, especially all the donors to my legal defense fund. Apparently, the money was well-spent. And I'd like to thank my very capable defense team: Phil Dubois (dubois@dubois.com) in Boulder, Ken Bass (kbass@venable.com) in Washington DC, Eben Moglen (em21@columbia.edu) in New York, Curt Karnow (karnow@cup.portal.com) in San Francisco, Tom Nolan in Palo Alto, and Bob Corn-Revere (rcr@dc1.hhlaw.com) in Washington DC. Most of the time they spent on the case was pro-bono. I'd also like to thank Joe Burton (joebur@aol.com) in San Francisco, counsel for the co-defendant.

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

1) William Bulkeley, "Cipher Probe", Wall Street Journal, Thursday 28 April 1994, front page.
2) John Cary, "Spy vs. Computer Nerd: The Fight Over Data Security", Business Week, 4 Oct 1993, page 43.

3) Jon Erickson, "Cryptography Fires Up the Feds", Dr. Dobb's Journal, December 1993, page 6.
4) John Markoff, "Federal Inquiry on Software Examines Privacy Programs", New York Times, Tuesday 21 Sep 1993, page C1.
5) Kurt Kleiner, "Punks and Privacy", Mother Jones Magazine, Jan/Feb 1994, page 17.
6) Steven Levy, "Battle of the Clipper Chip", New York Times Magazine, Sunday 12 Jun 1994, page 44.
7) Steven Levy, "Crypto Rebels", WIRED, May/Jun 1993, page 54.
8) Steven Levy, "The Encryption Wars: Is Privacy Good or Bad?", Newsweek, 24 April 1995, page 55.
9) John Markoff, "Cyberspace Under Lock and Key", New York Times, Sunday 13 Feb 1994.
10) Philip Elmer-DeWitt, "Who Should Keep the Keys", Time, 14 Mar 1994, page 90.
11) Vic Sussman, "Lost in Kafka Territory", U.S. News and World Report, 3 April 1995, page 32.
12) John Schwartz, "Can There Be Justice in Cyberspace?", Washington Post National Weekly Edition, 10-16 April 1995, page 32. Also, "Privacy Program: An On-line Weapon?", Washington Post, 3 April 1995, front page.
13) Jeff Ubois, "Hero or Villain? An Interview with PGP Creator Phil Zimmermann", Internet World, August 1995, page 78.
14) Ken Hoover, "Indictment Possible in Cyberspace Privacy Case", San Francisco Chronicle, 29 May 1995, page 6.
15) Paul Wallich, "Electronic Envelopes", Scientific American, Feb 1993, page 30.

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

Recommended Readings

Introductory Readings

1) Bruce Schneier, "Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd edition", John Wiley & Sons, 1995 (This book is the watershed work on the subject.)
2) Dorothy Denning, "Cryptography and Data Security", Addison-Wesley, Reading, MA 1982
3) Dorothy Denning, "Protecting Public Keys and Signature Keys", IEEE Computer, Feb 1983
4) Martin E. Hellman, "The Mathematics of Public-Key Cryptography," Scientific American, Aug 1979
5) Vic Sussman, "Taps for the code breakers", US News and World Report, 14 August 1995, page 16. (This article deals specifically with PGPfone.)
6) Steven Levy, "Pretty Good Phone Privacy", Newsweek, 28 August 1995, page 10. (This article also deals specifically with PGPfone.)
7) William Stallings, "Pretty Good Privacy", BYTE, July 1994, page 193

8) Philip Zimmermann, "The Official PGP User's Guide", MIT Press, 1994
ISBN 0-262-74017-6
9) Philip Zimmermann, "PGP Source Code and Internals", MIT Press, 1994
ISBN 0-262-24039-4
10) Simson Garfinkel, "PGP: Pretty Good Privacy", O'Reilly & Associates, 1994
ISBN 1-56592-098-8
11) William Stallings, "Protect Your Privacy - A Guide for PGP Users", Prentice-Hall, 1994
ISBN 0-13-185596-4

How to contact Zimmermann and Pretty Good Privacy Inc.

I prefer to be contacted by email in most cases. I rarely answer postal letters from end users. And I don't have the time to talk to many curious end users by phone. I don't do technical support over the phone anymore -- for that, send email messages to pgpfone-bugs@mit.edu. Fan mail is always welcome (to prz@pgp.com), but regretfully I don't always answer them. Until I move to California in August of 1996, you can reach me in Colorado at 303 541-0140 if you have any other business to discuss.

I used to make my living as a software consultant specializing in cryptography. But now I'm chairman and chief technology officer for my new company, Pretty Good Privacy Inc, so I don't have much time these days to do any consulting work. Pretty Good Privacy may be able to help you with your cryptographic needs. Pretty Good Privacy has custom and off-the-shelf versions of cryptography and authentication products available, such as public key implementations including the NIST DSS, RSA, Diffie-Hellman, and ElGamal, as well as custom product development services. You may also get the popular email encryption program PGP from Pretty Good Privacy, Inc. The company's address:

Pretty Good Privacy
555 Twin Dolphin Drive, Suite 570
Redwood City, California 94065
tel. (415) 631-1747
http://www.pgp.com

Computer-Related Political Groups

PGPfone is a very political piece of software. It seems appropriate to mention here some computer-related activist groups.

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

The Electronic Frontier Foundation (EFF) was founded in 1990 to assure freedom of expression in digital media, with a particular emphasis on applying the principles embodied in the US Constitution and the Bill of Rights to computer-based communication. They can be reached in San Francisco, at +1 415 668-7171. E-mail address: eff@eff.org.

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

The Voters Telecommunications Watch is a grassroots group chartered with encouraging public participation in the democratic process. Watching out for your civil liberties in the world of telecommunications, VTW alerts you to impending votes and other actions by the Federal Government that would negatively impact your civil liberties. You can learn more by reading their WWW page at URL http://www.vtw.org/ or by subscribing to vtw-announce at majordomo@vtw.org.

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

Reporting bugs in PGPfone

Send bug reports by email to pgpfone-bugs@mit.edu. Please be as specific as possible in your bug report to tell us how to reproduce the problem. Identify the exact model and configuration of your computer and modem, what version of the operating system, which version of PGPfone, and what settings you used, etc.

We have no telephone support for the freeware version of the product at this time, so you'll have to rely on email to report bugs. Please don't call me at home to have me talk you through some problem with PGPfone. When the commercial version of PGPfone becomes available, there will be a customer support phone line available from Pretty Good Privacy Inc.

The single biggest problem we have encountered during testing is modem quality. If the modem doesn't deliver clean reliable data without having the modem's error correction mode turned on, at the full speed required by the desired speech compressor, then PGPfone will exhibit poor performance. Please report details about your modem in any bug reports.

Where to get PGPfone

The Massachusetts Institute of Technology is the primary distributor of PGPfone, for distribution in the USA and Canada only. It is available from MIT's World Wide Web site at http://web.mit.edu/pgp, or http://web.mit.edu/network/pgpfone, a controlled Web site that has restrictions and limitations to comply with US export control requirements.

After each release of PGPfone from the MIT Web page, PGPfone will likely be downloaded from the MIT site to a great many BBS systems. If you don't have any local BBS phone numbers handy, here is a BBS you might try. The Catacombs BBS, operated by Mike Johnson in Longmont, Colorado, has PGPfone and PGP available for download by people in the US or Canada only. The BBS phone number is 303-772-1062. Mike Johnson's E-mail address is mpj@csn.org. Mike also has PGPfone and PGP available on an Internet FTP site for users in the US or Canada only; the site name is csn.org, in directory /mpj/, and you must read the README.MPJ file to get it. Remember to set mode to binary before doing an FTP download.

Technical Appendices

We are publishing a lot of technical information here in the appendices for those who are interested in digging deeper into PGPfone's architecture. Most other secure phone products, such as the STU-III or the AT&T phone products that use the Clipper chip, do not provide detailed information on their product's inner workings. We feel that if you are being asked to trust your confidential conversations to a secure phone product, you should be able to take a look under the hood and see how it works.

Appendix A -- Speech Compression Algorithms

PGPfone uses speech compression to allow speech to be reduced to a bandwidth that will fit over commonly available modems. The PGPfone architecture allows any speech compression algorithm to be negotiated between the two parties, as long as both parties support the same algorithm in their respective versions of PGPfone.

Speech compression engineers usually call a speech compression algorithm a speech or voice "coder", and the decompression algorithm a "decoder". These terms are not to be confused with "codes" that cryptographers use. Sometimes different disciplines have collisions in their jargon.

Speech compression is a very compute-intensive problem, if you want high-quality speech and a high compression ratio. The perfect speech coder would have these three properties: 1) Good speech quality, 2) high compression ratio, and 3) low computational load. Unfortunately, there are no known speech coders that can give you all three. Pick any two, and you can find a speech coder that fits. That is the trade-off.

Future versions of PGPfone will support a wide variety of speech coders, that will provide better compression ratios for more compute power, or less compute load for a lower compression ratio. For now, PGPfone version 1.0 supports the GSM speech compression algorithm for use over modems. GSM is the same compression algorithm used by European digital cellular phones. PGPfone also supports ADPCM compression for higher-bandwidth connections such as ISDN. ADPCM requires very little compute power to run, and sounds better than GSM, but it's only useful if you have a higher speed channel (32kbps) to transmit it.

Two variations of GSM are provided: standard GSM, and a variation of GSM that omits GSM's "long-term predictor". We call this latter variation "GSM lite". GSM lite has nearly the same speech quality as full GSM, but with less computing and a little less bandwidth required from the

modem. GSM is fairly compute-intensive, requiring a high-end Macintosh or a fast 486 or Pentium. PGPfone supports GSM and GSM lite with a range of sampling frequencies. Standard GSM was originally designed for telephones that take 8000 voice samples per second. We sample voice at various sampling rates, ranging through 4410, 6000, 7350, 8000, and 11025 samples per second. GSM lite at 7350 samples per second can be transmitted over a 14400 baud V.32bis modem. GSM at 11025 samples per second requires a V.34 modem. The faster you sample GSM, the better the voice quality, but this costs more compute load. GSM at 4410 samples/sec sounds really bad. At 11025 samples/sec, it sounds better than a normal phone assuming your computer audio hardware is also good.

GSM, like most voice coders, is asymmetrical in its compute load for compression and decompression. In other words, it takes much more computing to compress than it does to decompress. Maybe 2 or 3 times as long.

Another factor that might affect which voice coder you run is whether you are running full duplex (where both parties can speak at the same time, like on a normal telephone), or half duplex (where only one party can speak at one time, like on a CB radio. Over.). You may have a computer that can't run GSM at a nice high rate in both directions at once, but if you insist on full duplex, you may have to settle for lower speech quality and run GSM at a lower rate. If you want higher speech quality, you may have to settle for half duplex to get the better speech quality. Or buy a faster computer. For example, a Mac Powerbook 540c can encode and decode GSM at 7350 samples/sec at half duplex, but to run in full duplex, you'd have to encode GSM at perhaps 5512 samples/sec, but you could still decode GSM at 7350 samples/sec (because decoding is computationally cheaper than encoding). Which brings us to our next point.

PGPfone allows different voice coders to be run in different directions at the same time. This offers a lot of flexibility. Suppose you have a Mac Powerbook 540c, and you want to call someone running PGPfone on a PowerMac, which is a much faster computer. And let's say you want to run a full-duplex conversation. You can run a more compute-intensive higher quality voice coder on the PowerMac, and a less compute-intensive coder on the 540c (actually, the 540c can run the GSM coder at a nice 7350 samples/sec at half duplex, but we are assuming full duplex here, so the 540c will have to encode GSM at a lower rate). The 540c can decode and keep up with the better coder on the PowerMac because decoding is computationally cheaper than encoding. So the 540c user will hear better speech quality than the PowerMac user, because the PowerMac will be running the higher-quality voice coder. The PowerMac user will hear lower speech quality, because the 540c user will be running a lower-quality voice coder. Two PowerMacs with a pair of 28800 bps V.34 modems will be capable of running GSM at 11025 samples/sec in both directions, so both would hear excellent speech quality.

PGPfone also supports ADPCM compression, which delivers high sound quality, with low compute loads, but at a cost of a high bit rate of 32kbps. This is too fast for a V.34 28800 bps modem, but over an ISDN connection it should work fine. Or you could use it over the Internet

via a high speed connection such as Ethernet, if there is not too much traffic congestion. It sounds really great if you can get the bandwidth for it.

Appendix B -- Authentication and the "Man in the Middle" Attack

This section written by Colin Plumb.

When you establish a connection, PGPfone displays some nonsense words as "authentication parameters." What are they for? You've got an encrypted connection, right?

Yes, you do. But you don't know *with whom*. If Alice calls Bob, and can send data to Bob unmolested (even if an eavesdropper, Eve, is listening), PGPfone can set up a link that Eve can't listen in on—even though she heard all the nonsense that Alice and Bob's PGPfones said to each other.

But suppose that Eve is more resourceful and determined, and gets a bit more aggressive than just passively listening. Let's suppose that she inserts a computer of her own into the channel, so she can selectively alter the data that passes between Alice and Bob. She can insert data of her own, delete data, or pass data through unmodified. In this case, things can get more complex.

Alice calls Bob. Eve, however, gets in the way, and answers the call, then calls Bob herself. Alice's PGPfone sets up an encrypted connection with *Eve's* computer, and Eve sets up a separate encrypted connection with Bob's PGPfone. The two different encrypted connections use two different sets of keys. Eve then decrypts everything that Alice sends her, records it, then re-encrypts it to Bob and sends it on. And similarly in the other direction, from Bob to Alice. Alice and Bob both have established encrypted channels, and are talking to each other, but Eve is listening in.

The problem is that Alice and Bob aren't using the *same* encrypted channel, with the same set of keys. If it were a single, unbroken link, then Eve couldn't be listening in. But it's not—there is a "man in the middle."

That's where the authentication parameters come in. They are characteristic of the encrypted channel that has been set up, and the way that they are computed has been carefully designed so that Eve can't make them come out to any specific value. Or make the two come out to the same value. Eve can't even make them *tend* to come out to the same value. They're essentially two random values, and if Eve tries to break the link into two channels (with herself in the middle), the values will be different unless Eve is unbelievably lucky. The remaining 99.999+% of the time, Alice and Bob will notice that the parameters differ, so they know that someone is listening in the middle and can take appropriate action—like looking for wiretaps and hanging up or not discussing anything sensitive.

The assumption is that Eve can't fake Alice or Bob's voice well enough to splice in to the sound data going to Bob, an impersonation of Alice. The impersonation would have to read off the words that describe the Eve-Bob channel in place of Alice reading off the words that characterize

the Alice-Eve channel. Or vice versa. This appears to be a safe assumption for today, although as technology advances, this might become feasible, in which case PGPfone will have to switch to stronger authentication techniques than "that voice sounds like the right person."

Note that *because* using PGPfone's authentication parameters virtually guarantees that a man in the middle will be detected, it is very unlikely that anyone will ever attempt such an attack against you. But if you get lazy, and make a habit of not reading off the authentication parameters, and someone like Eve finds out, it becomes possible for them to mount this attack. It's a lot of trouble to set up, but unless you read off the authentication parameters, the attack *will* work.

Appendix C -- The PGPfone Protocols

This section written by Colin Plumb and Will Price.

This section is intended to help people evaluate the security of the system. For some parameters, the numeric values used are given to provide a concrete example, but be warned that the values used are not particularly important, and this document may lag behind changes to the code.

Also note that these protocols are designed to be very extensible and support downwardly-compatible extensions without breaking existing implementations. Despite this, all beta versions are considered potentially unfinished and may not be compatible with final versions. This is to reduce code bloat: if something in the beta version is found to be broken and in need of repair, compatibility with the brokenness will not be added to the release version.

During our discussions of protocols below, we will refer to our two communicating parties as Alice and Bob. The eavesdropper will be known as Eve.

General

PGPfone's protocols are designed to minimize, as much as possible, the information available to a passive or an active attacker. This includes, obviously, the payload data being sent, but also as much transactional information as possible that would be useful for traffic analysis. The identities of the parties (although that can often be guessed from the telephone numbers), how often they communicate, who else they communicate with, and so on.

There are a number of things—like deliberate delays in communication to disguise the speed of the computer being used—that are feasible for maximum protection against this sort of information leakage, but aren't implemented due to the perceived cost/benefit tradeoff. Still, it is designed so that these could be compatibly added.

There is one additional kind of attack that PGPfone is designed to be strong against: the physical, or rubber-hose attack. What if people break in and steal the contents of your computer? What if someone threatens you unless you tell them something? These are very powerful attacks which are difficult to defend against in general, but PGPfone does what it can. It does not leave anything lying around after a call which could be used to help break into a previously recorded encrypted call. It avoids making any record of its own of who you call, how often, how long you talked, and so on.

You still need to be careful that the PGPfone you use isn't tampered with. If someone were to replace your copy of PGPfone with one that sent its key out over the wire (say in the form of damaged packets which the legitimate receiver would ignore), an eavesdropper would easily be

able to decrypt a conversation.

Packet format

PGPfone provides its own link layer, sending packets across it, something like PPP. The big difference is that the data is encrypted. PGPfone uses two different formats for its packets depending on whether the call is over a packet switched network like the Internet or over a direct modem-to-modem connection.

Over a direct modem-to-modem connection, the basic packet format is similar to that of PPP, but not compatible. Packets are separated by special "delimiter" bytes. These have the value of 0x7E. Any occurrences of this byte inside packets is escaped using a mechanism described below, so each delimiter byte in the data stream marks a packet boundary. A single delimiter byte marks the end of the previous packet and the start of the next, although multiple delimiter bytes may be sent. The zero-length packet between them is ignored. This technique is commonly known as "byte stuffing". We do byte stuffing in modem-to-modem connections. But over the Internet, byte stuffing is skipped.

The basic packet format without byte stuffing consists of:

Packet type: 1 byte
Sequence number: 1 byte
Data: Variable length
CRC:
    Modem/AppleTalk: 2 bytes, or
    Internet: None, or 4 for Control packets on all connections

The packet type controls the interpretation of the data and distinguishes voice data, control packets, etc. Some types have subtypes, but that is described with the specific types. The sequence number wraps back to 0 once it reaches the maximum value for an unsigned byte. The various packet types are:

0	ACK
1	NAK (not currently used)
2	Voice
3	File (undefined data format, unimplemented)
4	Text (undefined data format, unimplemented)
5	Round Trip Timer
128	Control Reliable
129	Control Reliable, encrypted

The data portion of Voice packets is always encrypted if encryption has been negotiated.

Round Trip Timer packets are used to measure the round trip latency time between the two parties, to monitor performance. The format of this packet is a 1 byte flag representing which side originated this request, followed by a 4 byte millisecond timestamp on the sender's machine. The receiver should simply forward the packet back if it did not originate the packet.

For control packets, the CRC is 4 bytes long. Control packets are sent as reliable ACKed packets. We made the CRC longer for control packets in order to have greater assurance that the encryption setup packets were transmitted without error. It would be unnecessarily alarming to the user if non-matching authentication parameters falsely indicated a wiretap, when in fact it was just a line error.

There are various types of reliable control packets. To distinguish between these various types of control packets, the first data byte gives the subtype. Here we list the first byte of the packet data for the supported control packets:

0	Hello
1	DHHash
2	DHPublic
3	Info
4	Hangup
5	OpenVoice
6	VoiceSwitch
7	Talk
8	Listen

Call Setup

For call setup, each side sends four symmetric packets, plus (if all goes well) four ACK (acknowledgment) packets from each side. Each packet is sent only after the preceding packet is received from the other side. A special case occurs if the first packet received from either side indicates that encryption will not be used. In this case, packets two and three are skipped. The packets are defined as follows:

***Hello Packet*:** This is the first packet sent to begin a call. It is not encrypted. It is retransmitted every 2 seconds until an ACK is received or 30 seconds pass at which time hope is lost and the call is disconnected.

The contents of the Hello packet are organized into subpackets. The Hello packet begins with a Hello packet type byte. This is followed by the first subpacket, the second subpacket and so on. These subpackets may appear in any order except where otherwise noted. Each subpacket begins

with a type byte followed by a length byte followed by length number of bytes. The subtypes defined for the Hello packet in this version of the protocol are as follows:

0	Version
1	PublicName
2	HashAlgorithm
3	PrimeHashList
4	EncryptAlgorithms

Version - 1 byte equal to 1 for this version of the protocol. Future versions of software may transmit other values for this version byte. But these future versions generally should recognize and accept communications with older versions unless there is a good reason not to.

PublicName - Up to 32 bytes of identifying information about the sender. Can be used for a name, or other identifying information. This subpacket is completely optional on the part of the sender. This subpacket may appear either in the Hello packet, or in the Info packet that appears later in the protocol. Or it may be omitted from both. If it appears in both packets, it may have either the same name or a different name, depending on whether the sender wants to expose his real name in the unencrypted Hello packet. If the PublicName subpacket appears, it may be displayed by PGPfone to the user who receives it.

HashAlg - Currently defined as a 4 byte code "SHA1". Throughout this document, we refer to the NIST SHA-1 hash algorithm as either SHA or SHA-1. The intent is to use this field in the future should it ever become necessary to switch away from the SHA hash algorithm. Clients based on protocol version 1 should reject any call which does not specify SHA as the hash algorithm. In the future, a list of hash algorithms could be placed here, so clients should ignore any data beyond the first 4 bytes of this subpacket. If this subpacket is not sent, the default algorithm is SHA.

PrimeHashList - The first 8 bytes of this field are a salt. This is followed by an 8 byte hash for each prime allowed by the sender in order from the preferred prime upwards and then downwards. The salt is hashed together with the primes to obscure the primes used. If this subpacket is not sent, the key exchange is deactivated and the configuration should skip directly to the Info Packet. The receiver should choose the larger of the two parties' preferred primes, or the next-larger prime which both parties allow. The exact method for prime negotiation is described in a later section of this document.

The hashes are calculated in such a way that they specify more than just the prime field. They also include the key agreement algorithm in the hash calculation. PGPfone version 1.0 uses only plain Diffie-Hellman for its key agreement, but future versions may optionally use other algorithms such as elliptic curves. To calculate the hash, a key agreement algorithm identifier is concatenated with the prime and fed to the hash function. Of course, the salt is included with the hash as well.

EncryptAlgorithms - This is a list of 4 byte codes each of which represent an encryption algorithm that is enabled by that user. They should be presented in order of preference, with the most preferred appearing first. The first algorithm listed in the caller's list that also appears anywhere in the callee's list is agreed to as the algorithm to use.

The algorithms are identified by the following 4-byte codes:

"CAST"	CAST5 with 128-bit key (estimated key strength is 128 bits)
"TDEA"	Triple DES with 168-bit key (estimated key strength is 112 bits)
"BLOW"	Blowfish with 192-bit key (estimated key strength is unknown)
"NONE"	No encryption at all

If this subpacket is not sent, the call is unencrypted and the Diffie-Hellman is skipped.

If the receiver encounters an undefined subtype, the subpacket should be skipped.

Please note that all of the above subpackets are optional and each has a default value that should be assumed if the subpacket is not encountered. If no primes are agreed upon, then it results in an unencrypted call. A sender could theoretically send only the Hello packet type byte as a complete Hello packet. This would result in an unencrypted phone call and the packet sequence would then automatically skip to the Info packet.

***DHHash Packet*:** The DHHash packet begins with a DHHash packet type byte. This is followed by a full 20 byte SHA hash of the *g^x mod n* calculation using the Diffie-Hellman prime agreed upon in the Hello packet. This packet is important to require each party to fully commit to the Diffie-Hellman parameters being used before revealing the full calculation.

***DHPublic Packet*:** The DHPublic packet begins with a DHPublic packet type byte. This is followed by the full result of the Diffie-Hellman first half calculation. This is equal to the size of the prime. Using this, the receiver can calculate the shared secret and engage the encryption algorithms for all future packets.

If all three of the above packets are received and everything checks out, the protocol is now encrypted and unencrypted control or voice packets will be rejected.

***Info Packet*:** This packet can either be encrypted or unencrypted depending on whether an encrypted or unencrypted call has been negotiated. This packet is mandatory.

The contents of the Info packet are organized into subpackets. The Info packet begins with an Info packet type byte. This is followed by the first subpacket, the second subpacket and so on. These subpackets may appear in any order except where otherwise noted. Each subpacket begins with a type byte followed by a length byte followed by length number of bytes. The subtypes defined for

the Info packet in this version of the protocol are as follows:

5	VoiceEncoders
6	VoiceDecoders
7	Duplex
8	Sync
9	AuthenticationBytes
10	WordList
11	PublicName

VoiceEncoders - This is a list of 4 byte codes representing the speech compression algorithms that the sender is capable of encoding with.

VoiceDecoders - This is a list of 4 byte codes representing the speech compression algorithms in order of preference that the sender would like to decode with.

Duplex - This is a 1-byte packet equal to either 1 or 0. If equal to 1 on both sides, a full duplex call is negotiated, otherwise half duplex is used.

Sync - If using direct modem connections and a synchronous connection is desired, this 1-byte field may be set to 1 to negotiate synchronous mode on both sides. If either party sets this byte to 0, then the modem connection will run in asynchronous mode.

AuthenticationBytes - This 1-byte field contains the maximum number of authentication bytes. This must be no greater than 4 and no less than 2. The lower number of the two sides should be presented to the user.

WordList - This 1-byte field is set to 1 if the biometric authentication bytes should be translated into the biometric word list (see appendix). If this byte is set to 0 by either side, the authentication bytes will be displayed in hexadecimal.

PublicName - Up to 32 bytes of identifying information about the sender. Can be used for a name, or other identifying information. This subpacket is completely optional on the part of the sender. This subpacket may appear either in the Hello packet (which is always unencrypted), or in the Info packet (which is encrypted, if this is an encrypted call). Or it may be omitted from both. If it appears in both packets, it may have either the same name or a different name, depending on whether the sender wants to expose his real name in the unencrypted Hello packet. If the PublicName subpacket appears, it may be displayed by PGPfone to the user who receives it.

At this point in the protocol after all four of these packets have been received and ACKed, the protocol enters the call stage. The voice coders are already set by the Info packet, so the sound

modules are directed to begin sending packets according to the duplex rules negotiated. If encryption was negotiated, the data in all packets from the Info packet on are encrypted. Further control packets may be sent after this point to turn the channel around in half duplex, switch coders, or to hangup. These are encrypted packets which are ACKed just like the above setup packets. If they are not ACKed, they are resent every 2 seconds.

Once the voice traffic begins, a stream of voice packets are sent. PGPfone uses slightly different voice packet formats for modem-to-modem protocol versus Internet protocol. Internet protocol for voice must be designed to contend with unpredictable timing delays in packet transmission, depending on network congestion. To address these problems, a future version of PGPfone will use a packet format similar to standard Real Time Protocol (RTP), which is fully defined in Internet RFC 1889. The current version of PGPfone does not have RTP's timing control features, which would improve the handling of unpredictable Internet packet delays.

Here is the packet definition for PGPfone voice packets sent over the Internet.

Type	1 byte
Sequence	1 byte
Data	<= 1024 bytes
CRC	4 bytes (reliable ACKed packets only)

The type byte and the sequence number byte are present in all packet types, voice or not. The type byte distinguishes between control packets and voice packets. The sequence number byte is used to detect packets arriving out of sequence and is also used to help synchronize the block cipher, which runs in counter mode.

Standard RTP uses cipher block chaining (CBC) mode for encryption, while PGPfone uses counter mode for encryption. There are good reasons why we decided to use counter mode, as explained later.

Internet Extensions for Call Setup

A simple protocol for handling calls over the Internet is used for initiating calls before the main PGPfone protocol is invoked at a higher layer. All PGPfone Internet packets are sent as UDP packets over port 4747 (decimal). The types of call setup packets are:

1 Call
2 Busy
3 Probe
4 ProbeResponse

5	Reserved
6	Reserved
128	Accept

To initiate a call, a Call packet is sent. The format of this packet is the packet type byte followed by one byte for length, and then the optional public name of the caller of the specified length. The public name is only included if the user of PGPfone has checked the "Unencrypted" and "Outgoing" checkboxes in the Phone dialog.

When a remote PGPfone receives a call packet, it immediately sends back a ProbeResponse packet which has exactly the same format as a Call packet except for the packet type byte. The ProbeResponse allows the caller to know that the address is correctly identified even if the called party does not answer. The ProbeResponse packet is optional.

After the called party decides to accept a call, an Accept packet is sent. The format of the Accept packet is simply a one byte packet type. The Accept packet might come long after a Call packet, because a user response to accept the call is usually what triggers an Accept packet.

At any time, a Probe packet can be sent to interrogate an IP address in an attempt to get a ProbeResponse. This could be used for instance to scan a range of IP addresses for a specific user without ringing everyone's phone. Receiving a Probe packet should not result in any visible indication to the user being probed. This protocol feature is useful because some users do not have static IP addresses, but rather their addresses change (within a range) every time they attach themselves to the Internet. The Probe packet allows you to search for those users that have floating IP addresses. Future versions of PGPfone will exploit this.

Encryption

When a call is first established, packets are sent across unencrypted while negotiation of the encryption parameters proceeds. However, as early as possible, the stream is encrypted to deny eavesdroppers information about what's going on.

The data in the PGPfone packets is encrypted in counter mode using one of the three ciphers: TripleDES, CAST, or Blowfish. Counter mode offers several advantages for this application. Because it does not actually encrypt or decrypt any of the original data, the counter encryption and decryption can be pre-computed for each packet, requiring only an XOR once the data becomes available. This allows us to further reduce latency in future implementations by pre-computing the encrypted keystream during interpacket gaps before the voice data becomes available for encryption or decryption.

Like CFB, counter mode can encrypt any byte length of data without padding. Even though the

cipher is used on 8 byte blocks, the result is XOR'd only up to the length of the data.

The 8 byte counter used by PGPfone's cipher is structured as follows. Note that all integer subfields are treated as big-endian (high byte first).

Byte 1	Packet type
Bytes 2-6	Packet counter, mod 2^40
Bytes 7-8	Cipher block counter

The low bits of the packet counter field are kept in sync with the packet sequence number, which is sent in the clear at the beginning of each packet. The full 5-byte packet counter is set to 0 at the beginning of each call, and is incremented by 1 with each packet until it rolls over at 2^40 packets. One of the principles of counter mode is that the counter should never repeat with the same key. We doubt anybody will be having any calls with more than 2^40 packets. At an average of 6 voice packets per second, a phone call comprised of 2^40 packets would last 5809 years. A call comprised of merely 2^32 packets would last 22 years.

The packet type further differentiates the counter by making each packet type unique. There are two general types of encrypted packets that PGPfone sends: control and voice. This field is simply a duplicate of the packet type byte at the beginning of the packet. This allows independent packet sequence numbers for control packets and voice packets.

The cipher block subfield is simply a counter calculated per packet starting at 0 and incrementing by 1 for each 8-byte cipher block within the current packet's data. It is reset to 0 at the beginning of each packet.

This elaborate method of using a structured counter in counter mode offers important advantages over the traditional method of simply incrementing a 64-bit counter. It allows us to re-establish cipher sync at the start of each packet, by using the packet sequence number as part of the counter. This makes the protocol more resilient to transmission errors that might otherwise cause us to lose cipher sync if we used a simple continuously incrementing 64-bit counter.

Diffie-Hellman prime negotiation

The Diffie-Hellman key agreement protocol requires that both parties agree upon a generator and a prime. PGPfone uses a fixed generator of 2, which has speed advantages. The prime is agreed upon somewhat interestingly, however.

The Diffie-Hellman protocol relies on the difficulty of extracting discrete logarithms modulo a prime. This is difficult for almost all primes, but a deliberate search can produce some “cooked”

primes for which the problem is much easier. Obviously, you want to avoid using such primes.

However, there is no particular requirement to keep the prime secret or change it frequently. So PGPfone assumes that each party generates all the primes that it will use by a common algorithm, beforehand. This way, each party can know that the primes aren't cooked—if I made the prime and didn't cook it, it's not cooked.

The main reason for having multiple primes is to provide some room for security/speed tradeoffs.

However, some people might like to use "private" primes. These primes are never disclosed on a channel. This incidentally makes an attacker's problem even more difficult. But the main reason why we don't transmit the primes themselves over the channel is that we assume the other party can't check the goodness of the prime on the fly, so we just assume he already has the prime in advance and has already assured himself that it is a good prime to work with by making such decisions off-line. (For the initial software release, adding primes requires changing the source code, but that may change in the future.) PGPfone lets two parties figure out which primes they have in common by exchanging hashes of the primes.

However, such a scheme still reveals to anyone listening to the channel some sort of identification code (the hash) of the unknown primes. If you have a private prime and I have the same private prime, someone listening in could infer that we have close connections even if we don't call each other regularly.

To help prevent this sort of traffic analysis, PGPfone adds "salts" to the hashes of the primes that are sent. Each party chooses a random 8-byte value and hashes that in with each prime. This salt is sent in the clear along with the hashes of the primes.

You can use the salt to hash the primes that you have and compare the hash with the list I sent you. If you find a match, I have that prime, too. I can do the same with the salt and hashes you send me. The protocol on the wire is simple: I send you my salt and hashes, and you send me yours in reply. We may both request at the same time and then send a redundant reply, but that is harmless.

All that remains is to choose the prime, from among those that we have discovered that we have in common, that we shall use. The algorithm is simple: each of us sends the hashes of our primes in descending order of preference. Thus, the prime whose hash is listed first is the most preferred, the prime listed second is the next most preferred, etc. Each of us can find, among the primes that we both support, which one I most prefer (listed first on my list) and which one you most prefer (listed first on your list). Then we need only choose between the two. For the sake of improved security, the choice is defined as the numerically larger.

An eavesdropper can determine which of the primes each of us has advertised he also knows, and which he does not, but is very limited in what he can learn about the primes that he does not know

and the prime that we eventually use.

If he can determine which primes are in each of our preference lists up to the first prime each of us lists that the other has, he can determine which prime we use. Of course, he can determine from later Diffie-Hellman use the size of the prime, which lets him exclude many primes from the list of possibilities. If he assumes that we don't share any of the primes he doesn't know, he can determine the size of the prime that we would agree upon, and if we use a prime of a different size, he knows that we share a private prime.

This tells an attacker something about our association (that we know each other fairly well and are paranoid enough to generate a private prime), but it's still pretty limited, and if the preference list always lists the public primes after the private primes of a given size, an attacker will be unable to determine which one we have used.

But this only applies to a passive attacker. Until PGPfone supports private primes better, everyone will have the same list of primes and this whole issue is moot, but there are some active attacks that would allow an active wiretapper to discover that Alice and Bob share a private prime. We won't describe the attacks here. Merely finding out that Alice and Bob share a private prime seems to be of such limited value to an attacker that it seems not worthwhile to defend against such attacks.

Diffie-Hellman Key Agreement

Once the parties have already agreed upon a Diffie-Hellman prime to use, and our standard generator of 2 (for speed), then Bob, who we assume is initiating this, chooses a random secret x_b, computes $y_b = 2^{x_b} \pmod{p}$, and hashes y_b with SHA. This hash is sent to Alice. For the purposes of hashing, y_b is represented as a big-endian array of bytes, the array just large enough to hold the prime p.

When Alice receives this packet, she chooses a random x_a, computes $y_a = g^{x_a} \pmod{p}$, hashes y_a in the same way and sends this hash to Bob. Bob replies by sending his y_b, and Alice checks that the hash matches. Alice sends her y_a, and Bob checks that the hash matches.

Assuming all goes well, both parties compute the shared secret $z = y_a^{x_b} = (g^{x_a})^{x_b} = g^{x_a \cdot x_b} = (g^{x_b})^{x_a} = y_b^{x_a}$ and use that for a shared secret. Actually, both parties discard the most significant 4 bits of the result and round down to the nearest byte. The remaining bytes, most to least significant, are the shared secret.

This "derating" procedure is done to avoid biases in the shared secret caused by operations modulo the prime p. For example, if the most significant byte of the prime were 128, then the corresponding byte of z would be between 0 and 127 most of the time, and occasionally 128, but

never any higher. Thus, the most significant bit is quite non-random. The derating is done based on the size of the prime (where 127 is the last 7-bit number and 128 is the first 8-bit number, etc.), which is decreased by 4 and rounded down to the next multiple of 8 bits. In the common case, where primes are a multiple of 8 bits long, this amounts to throwing away the most significant byte.

In addition to agreeing on a shared secret, the result is hashed for authentication purposes. The bytes hashed for authentication purposes are formatted as follows:

- The prime modulus, in the form of a big-endian 2-byte count of bytes (leading zeros suppressed), followed by a big-endian array of bytes containing the prime.
- The generator (2), in the same format.
- The shared secret z, all of it (not derated), in the same format.

Note that the byte count and number of bytes may be less than the byte count of the prime modulus.

The purpose of exchanging the hashes of the Diffie-Hellman public parameters before exchanging the parameters themselves is to force each side to commit to their parameter before it is told the other side's parameter. This makes any attack which attempts to produce a given hash a daring attack. If this commitment were not there, Alice could, knowing Bob's yb, attempt to choose her xa so that the hash of the shared secret has certain properties.

This is particularly significant when an eavesdropper, Eve, is engaging in an active attack by trying to play "man in the middle," talking to both Alice and Bob but selectively forwarding packets so that each thinks they are talking to the other directly. Eve could enter into Diffie-Hellman negotiation with both Alice and Bob and contrive to request a cloak change first, so both Alice and Bob would reply with Diffie-Hellman parameters. Alice and Bob will later compare the hashes of the Diffie-Hellman parameters to see if they have a secure channel. If it were not for the initial hash, "committing" each party to their number before receiving the other's number, Eve could attempt to choose the numbers she sends to Alice and Bob so the hashes are similar or the same.

Any such attack is referred to as a "safe" attack. Eve could know, before doing something Alice and Bob would notice, that her attack would succeed.

But because this is impossible, this is a "daring" attack. Eve must start her attack, and risk detection, before she can know if it will succeed. Because she has no control over the hashes at all, the hashes used for authentication can be considerably reduced in size. They are just random numbers, and unless Eve can generate a large pile of x values such that the y values all have the same hash (which seems to involve both breaking SHA quite thoroughly and solving a lot of discrete logarithm problems), and then try to do a birthday attack as described on the hash of the whole Diffie-Hellman authentication information, you only need enough hash to ensure that Eve cannot usefully rely on an accidental similarity.

This provides an argument that even one byte of hash is enough! Consider that you wish to do something covertly. There are two chances to consider: the chance of success, and the chance of detection.

If the former is low, you may be able to keep trying until you succeed as long as the latter is even lower. But if the latter is high, it's pretty crazy to start trying, because you're going to get caught. With a 1-byte hash, the chance of success is 1/256, less than 0.5%, while the chance of detection is 255/256, greater than 99.5%. You can't try and give up without attracting some attention.

Actually, there is one possibility. Eve could try, and if she notices that she is about to get caught, she could block the following authentication by simulating line noise and dropping the connection or otherwise emulating a glitch. If she could do this 255 times, she'd expect to break into one call. She'd have to convince Alice and Bob that the line between them was very bad, and let some calls go through without attempted interception so Alice and Bob wouldn't give up, and she'd end up with a very few calls.

This is noticed by the communicating parties, but in this situation, Eve has a very good chance of avoiding blame (especially when Alice and Bob are using beta software). Thus, PGPfone uses a somewhat larger hash, but it still doesn't need very much.

Authentication

There is currently only one protocol for PGPfone authentication: biometric. Each side computes a hash of the authentication parameters (using SHA for now, but this is a negotiated option), and displays the first few bytes (the number of bytes is also a negotiated option, currently 4).

The reason that only using 4 bytes of hash is safe, while 16 or 20 bytes are computed by common hash algorithms, is discussed above where the daringness of an attack is described. PGPfone ensures that no party to the call ever has an opportunity to try numbers until one is found that produces a given authentication hash. Because PGPfone uses a protocol that prevents such attacks, only two to four bytes of hash material needs to be authenticated by voice.

The authentication consists of hearing the person you want to talk to read off the authentication code that PGPfone displays, or hearing them agree that the words you just read off are correct. If it's their voice (and their, untampered, copy of PGPfone), then they are the only other person listening to what you're saying. It's suggested that one person read off the first half of the authentication code and the other respond with the other half.

There are two formats for the display. The fallback is hexadecimal, but the standard (used unless

option negotiation reveals that one of the parties does not support it) involves two 256-word lists, chosen so that any entry in each list is phonetically distinct from all other entries. This is to minimize confusion over what was said. The lists consist of two-syllable and three-syllable words, and are used alternately. The alternation provides an easy way to detect errors in long sequences of authentication words, so that these word lists can be used for reading lengthy key material over a phone line in other cryptographic applications. However, PGPfone only uses short sequences of two to four words.

This authentication is based on the assumption that real-time mimicking of a person's voice currently has too high a chance of being detected. As such an attack becomes more practical, stronger authentication may have to be added, using digital signatures.

A simple technique for making this biometric approach stronger is to communicate the authentication parameter words in a manner which includes some shared background with the other person. If the word is "cellulose," remind the other person about a story to do with printing, logging, the time you were lost in the forest, or something like that. If it's "gizzard," tell a food story. And so on.

This is much harder for an automated system to fake, since they don't know you. A skilled impersonator with a thorough background dossier might manage, but if you manage to attract that level of effort to intercepting your conversation, then thank you. That's effort the attacker can't spare listening to the rest of us.

Appendix D -- How PGPfone's DH Primes Were Derived

This section written by Colin Plumb.

PGPfone uses Diffie-Hellman key exchange to decide on a session key for communication. Diffie-Hellman requires a prime modulus to work with. (Note that this is unlike RSA, which uses a modulus which is the product of two primes.) This note explains how these primes are computed.

"Kosherized" Primes

The security of the Diffie-Hellman exponential key agreement depends on the difficulty of computing discrete logarithms. It has been suggested that it may be possible to contrive a prime modulus such that it is easier to compute discrete logs in that modulus's field.

This is one reason why some have suggested that rather than trust a prime generated by someone else, it is better to generate your own prime. This approach can lead to a massive proliferation of primes in use in a community of Diffie-Hellman users. But there are advantages in having everyone use a small common set of well-known public primes. It makes key management easier, and allows precomputation of the first phase of the Diffie-Hellman exchange, which speeds things up.

The Diffie-Hellman primes used by PGPfone were derived in such a way that anyone may independently verify that the primes were not contrived to make it any easier to compute discrete logs. Thus, there is no reason why everyone can't trust the prime numbers selected for PGPfone. We will describe here exactly how these primes were derived.

The prime generation is a variant of the technique for DSS prime generation suggested by David Kravitz (of NSA, at that time) and recommended by NIST. It is based on SHA and an arbitrary seed. Kravitz calls this process "kosherizing the primes".

SHA is the NIST standard Secure Hash Algorithm with a 160-bit output. It is widely believed in cryptographic circles that it is computationally infeasible to choose an input value for SHA that will produce some chosen output hash. We refer to the hash algorithm throughout this document as SHA, even though it is actually the NIST-revised SHA, known as SHA-1.

Careful prime generation is significant to Diffie-Hellman because the security of Diffie-Hellman key exchange rests on the difficulty of the discrete log problem: given y, g and p, find x such that y = g^x mod p. This is, in general, very difficult.

However, there are certain "cooked" primes p such that this is comparatively easy to solve. These

"cooked" primes are very rare - given a random prime p, the chance that it is cooked is so small as to not be worth worrying about. So to choose a prime and be convinced that it is not cooked, it suffices to choose a prime from a given range in a manner that prevents anyone from forcing the choice to one of the rare "cooked" values.

This is where SHA comes in. By using SHA to generate the primes, it is possible to show the impossibility of steering the prime-selection process to one of the rare "cooked" values, since it's not possible to steer the output of SHA.

Since SHA generates 160-bit outputs, and we want larger primes (such as 1024 bits), we need to use SHA multiple times. The first invocation of SHA generates the low 160 bits of the number. The next invocation generates the next lowest 160 bits, and so on, until enough bits are available. For example, a number of up to 1024 bits can be generated as follows:

(SHA(seed1) + 2^160 * SHA(seed2) + 2^320 * SHA(seed3) + 2^480 * SHA(seed4) + 2^640 * SHA(seed6) + 2^800 * SHA(seed7) + 2^960 * SHA(seed8)) mod 2^1024

The sum inside the parentheses generates a number 7*160 = 1120 bits long. The modulo operation throws away the extra 96 bits that aren't needed. The only thing needed here is 7 seed values. What the seed values are is unimportant, as long as they are published so that anyone may verify the fact that the number is generated by SHA.

(The 20-byte SHA output is treated as a big-endian number for the purpose of these computations.)

For PGPfone, the seed1 value is an ASCII string of a recognizable phrase. seed2 is created by incrementing the last byte of the phrase by 1. seed3 is created from seed2 by incrementing the byte again, et cetera.

PGPfone uses 5 user-selectable primes derived from the seed, of different sizes: 512, 768, 1024, 1536, and 2048 bits. For the initial release of PGPfone, the seed phrase is an ASCII string containing a quote from Mahatma Gandhi: *"Whatever you do will be insignificant, but it is very important that you do it."*

After the number is created, a few changes are made to it:

- The most significant bit is set, to ensure that the number is really 1024 bits long (or however many bits are desired).
- The second most significant bit is set to make the number larger. The effort of performing Diffie-Hellman computations depends almost exclusively on the number of significant bits in the number. There is no real difference between high 1024-bit numbers close to 2^1024-1 and low 1024-bit numbers close to 2^1023. The effort of solving the discrete log problem, however, depends more heavily on the value of the number, so picking primes from the high half of the 1024-bit range provides slightly better security at no cost.

Then this number is used as a starting value for a sequential prime search. The first suitable prime greater than or equal to the starting number is the generated prime.

"Suitable prime" is stronger than just "prime". For a prime p to be suitable, (p-1)/2 must also be prime. If this is true, then the generator g in the Diffie-Hellman computation can be chosen to be 2 without any loss of security. This choice speeds up computation with the prime, so as primes are generated rarely but used often, the extra effort taken to find a "suitable" prime is amply repaid.

Comparison with Kravitz's technique

David Kravitz originally designed his technique for generation of primes for the NIST Digital Signature Standard (DSS). DSS requires two primes, so seed1 and seed2 were used to generate the small prime, and the large prime was generated using seed3 and upwards. Since we only have one prime, we start with seed1.

There was also a modular relationship required between the smaller and the larger prime, which Kravitz' technique enforced by decreasing the large prime to make the relationship hold. That does not apply to this operation and is omitted.

The seed1, seed2, etc. generation using incrementing is exactly as in Kravitz's original design.

Kravitz did not set the second highest bit. That is a simple modification to make things slightly more difficult for an attacker.

The most significant difference is that Kravitz didn't generate a starting value and do a sequential search for a prime; his technique kept using more seed values to generate more random numbers, which were then tested for primality.

His technique has some mathematical elegance advantages, in particular a sequential search produces some primes (those with a long gap between them and the previous suitable prime) more often than others (those which closely follow the previous suitable prime), but the change is quite minor, and definitely not significant for Diffie-Hellman primes. (Diffie-Hellman primes aren't secret, so it isn't important to pick as randomly as possible to make guessing difficult.) The advantage is that sequential numbers can be tested for primality much faster than random numbers. Prime generation is slow enough already; it doesn't need to be made any slower.

Implementation details

This section describes the implementation of the search, for those who wish to follow the supplied

code and verify that it performs the operations described above.

Generating the initial random number is straightforward enough. Searching for suitable primes, however, is the subject of a great deal of optimization. The prime search proceeds in two steps: sieving and testing.

First note that we are looking for a prime p such that q = (p-1)/2 is prime. Thus, q must be odd and p must be congruent to 3 mod 4.

Further, neither q nor p = 2*q+1 may be congruent to 0 modulo 3. The constraint on p means that q may not be congruent to 1 modulo 3, so q must be congruent to 2 modulo 3. Combining this with the fact that q must be odd implies that q must be congruent to 5 modulo 6. p must, therefore, be congruent to 11 modulo 12.

The first step is to take the initial number s, find (s-1)/2, and round it up until it is congruent to 5 modulo 6. This number is n1. Searching for q proceeds in steps of 6 from this base. n2 = 2*n1+1 is the base for the search for p, which proceeds in steps of 12.

To perform the sieving, a large bit array (65536 bits, or 8192 bytes) is allocated. Bit i of the array will be cleared if either of n1+6*i or n2+12*i have small divisors.

The small divisor test is performed by initially setting the array to all ones, and then for each prime divisor d up to 65536 (since we are stepping by 6 = 2*3, 2 and 3 are excluded), n1 modulo d is computed, and the remainder used in a bit of computation to find the first index i such that n1+6*i is divisible by d. Then every dth bit from that point onwards in the array is cleared. Bits for which n2+12*i is divisible by d are cleared similarly. n2 modulo d can be computed from n1 modulo d, by doubling and adding 1, so the computation is very easy.

After that, the bit array is walked through looking for set bits, which indicate possible primes q and p. At least, each candidate c1 has no small prime factors, and c2 = 2*c1+1 has no small prime factors. For each candidate pair, a Fermat test to the base 2 of c2 is performed. i.e. we check if 2^(c2-1) == 1 (mod c2). If not, a dot is printed and the next candidates are sought.

If we reach the end of the array, the starting number n1 is increased by 6*65536 and the process repeats. The search never gives up. A slash (/) is printed when the bit array is exhausted like this.

The exponentiation 2^x (mod m) is handled in specially optimized code that takes advantage of the simplifications allowed by using 2 as a base.

If the first Fermat test is passed, a "+" or "-" is printed, depending on some slightly arcane mathematical property. (It's the sign of 2^c1 mod c2, which is +1 or -1 if the test is passed.) Then a second Fermat test is performed, this time on c1, to see if 2^(c1-1) == 1 (mod c1). If not,

a dot is printed and the search resumes.

If the second test is passed, we have found a suitable prime. An asterisk (*) is printed to indicate success. But since the primality tests have a (ridiculously low) chance of failing, some extra "confirmation" tests on both c1 and c2 are performed. An asterisk is printed after each of these is completed, too. Three confirmation tests are performed on each of c1 and c2, so a total of seven asterisks are printed.

(In the case that a confirmation test fails, a dot is printed and the search is resumed, but this is not going to happen in your lifetime.)

At the end of this, c2 is the suitable prime p that is sought.

Appendix E -- Biometric Word Lists

PGPfone uses a special list of words to authenticate the Diffie-Hellman key exchange, via biometric signatures. The word list serves the same purpose as the military alphabet, which is used to transmit letters over a noisy radio voice channel. But the military alphabet has 26 words, each word representing one letter. For our purposes, our list has 256 carefully selected phonetically distinct words to represent the 256 possible byte values of 0 to 255.

We created a word list for reading binary information over the phone, with each word representing a different byte value. We tried to design the word list to be useful for a variety of applications. The first application we had envisioned was to read PGP public key fingerprints over the phone to authenticate the public key. In that case, the fingerprint is at least 16 bytes long, requiring at least 16 words to be read aloud. When a new version of PGP uses SHA instead of MD5 for its key fingerprints, it requires 20 bytes to be read. Experience has shown it to be fairly tedious and error prone to read that many bytes in hexadecimal, so it seems worth using a word list to represent each byte by a word.

Some applications may require transmitting even lengthier byte sequences over the phone, for example, entire keys or signatures. This may entail reading more than a hundred bytes. Using words instead of hex bytes seems even more justified in that case.

When reading long sequences of bytes aloud, errors may creep in. The kinds of error syndromes you get on human-spoken data are different than they are for transmitting data through a modem. Modem errors usually involve flipped bits from line noise. Error detection methods for modems usually involve CRCs to be added, which are optimized for detecting line noise bursts. However, random sequences of spoken human words usually involves one of three kinds of errors: 1) transposition of two consecutive words, 2) duplicate words, or 3) omitted words. If we are to design an error detection scheme for this kind of data transmission channel, we should make one that is optimized for these three kinds of errors. Zhahai Stewart suggested a good scheme (in personal conversation with me in 1991) for error detection of these errors.

Stewart's scheme for error detection while reading aloud long sequences of bytes via a word list entails using not one, but two lists of words. Each list contains 256 phonetically distinct words, each word representing a different byte value between 0 and 255. The two lists are used alternately for the even-offset bytes and the odd-offset bytes in the byte sequence. For example, the first byte (offset 0 in the sequence) is used to select a word from the even list. The byte at offset 1 is used to select a byte from the odd list. The byte at offset 2 selects a word from the even list again, and the byte at offset 3 selects from the odd list again. Each byte value is actually represented by two different words, depending on whether that byte appears at an even or an odd offset from the beginning of the byte sequence.

For example, suppose the word "adult" and the word "amulet" each appears in the same corresponding position in the two word lists, position 5. That means that the repeating 3-byte sequence 05 05 05 is represented by the 3-word sequence "adult, amulet, adult".

This approach makes it easy to detect all three kinds of common errors in spoken data streams: transposition, duplication, and omission. A transposition will result in two consecutive words from the even list followed by two consecutive words from the odd list (or the other way around). A duplication will be detected by two consecutive duplicate words, a condition that cannot occur in a normal sequence. An omission will be detected by two consecutive words drawn from the same list.

To facilitate the immediate and obvious detection by a human of any of the three error syndromes described above, without computer assistance, we made the two lists have one obviously different property: The even list contains only two-syllable words, while the odd list contains only three-syllable words. That suggestion came from Patrick Juola.

Now as it so happens, due to some refinements we made in our key exchange protocol, PGPfone does not need to have a human read any long sequences of bytes aloud. No more than four bytes are read aloud for biometric signatures to authenticate the Diffie-Hellman key exchange. In fact, two bytes would suffice, but we used four bytes in our lab tests, and forgot to change it to two bytes before the release. This means that errors of the above types are unlikely. So we could have used a single 256-word list instead of two 256-word lists. Nevertheless, we had already started to develop the word lists for working with long byte sequences for multiple applications, such as PGP. So it seemed worthwhile to just go ahead and use the full two-list scheme with its fancy error detection properties, even if two lists seem to be a bit of an overkill.

The idea behind building the word lists was to develop a metric to measure the phonetic distance between two words, then use that as a goodness measure to develop a full list. Grady Ward provided us with a large collection of words and their pronunciations, and Patrick Juola used genetic algorithms to evolve the best subset of Ward's list. To briefly summarize what he did, he made a large population of guesses and let the population "sexually reproduce" by exchanging words with other guesses -- and, like biological evolution, the better guesses survived into the next generation. After about 200 generations, the list had mostly stabilized into a best guess, with far greater phonetic distance between the words than what we started with in the initial guess lists.

The first major hurdle was the development of the metric. Linguists have studied sound production and perception for decades, and there is a standard feature set used to describe sounds in English. For example, say the words "pun," "fun," "dun," and "gun" (go ahead, try it), and notice how your tongue keeps moving back in your mouth on each word. Linguists call this the "place of articulation," and noises that are very different in this feature sound different to English speakers. Combining the features of all the sounds in a word gives us a representation of the sound of the entire word -- and we can compute the phonetic distance between a pair of words.

Actually, it wasn't that simple. We didn't know how to weight the various features, certain word-level features like accents were hard to represent, and the feature-based analysis simply fails for certain sounds. There were also a few other more subtle criteria; for example, we wanted the words to be common enough to be universally recognizable, but not so common as to be boring -- and we didn't want confusing words like "repeat" or "begin" or "error". Some sound features are less perceptible to non-native-English speakers, for example, some Japanese speakers might hear and pronounce "r" and "l" the same way. It would be nice if the words were short enough that you could fit enough of them on a small LCD display. Large consonant clusters ("corkscrew" has five pronounced consonants in a row) are sometimes hard to say, especially to non-English speakers. One way or another, we tried to incorporate all these criteria into a filter on the initial dictionary list or into the distance metric itself.

After the computer evolved the winning list, we looked at it. Yes, the words were phonetically distinct. But many of them looked like a computer picked them, not a human. A lot of them were just ugly and dumb. Some were repugnant (the computer included the phonetically distinct word "nigger" from the dictionary), and some were bland and wimpy. So we applied some "wetware" augmentation to the list. Some words were deleted, and replaced by some human-chosen words. We had the computer check the new words against the list to see if they were phonetically distant from the rest of the list. We also tried to make the words not come too close to colliding phonetically with the other words in the larger dictionary, just so that they would not be mistaken for other words not on the list.

There were a variety of selection criteria that Juola used in his algorithms. He may publish a paper on it that goes into more detail. This appendix was just a brief overview of how we built the list.

I'm not entirely happy with the word list. I wish it had more cool words in it, and less bland words. I like words like "Aztec" and "Capricorn", and the words in the standard military alphabet. Maybe we should revise the word list during the beta test phase, and publish a new list in the full release version of PGPfone. If you have any suggested words you'd like to see added or deleted, send them in to pgpfone-bugs@mit.edu, and while you're at it, send a copy to Patrick Juola at patrick.juola@psy.ox.ac.uk.

Here are the full word lists, both odd and even.

Two Syllable Word List

aardvark
absurd
accrue
acme
adrift
adult
afflict
ahead
aimless
Algol
allow
alone
ammo
ancient
apple
artist
assume
Athens
atlas
Aztec
baboon
backfield
backward
basalt
beaming
bedlamp
beehive
beeswax
befriend
Belfast
berserk
billiard
bison
blackjack
blockade
blowtorch
bluebird
bombast
bookshelf
brackish
breadline
breakup
brickyard
briefcase
Burbank
button
buzzard
cement
chairlift
chatter
checkup
chisel
choking
chopper
Christmas
clamshell
classic
classroom
cleanup
clockwork
cobra
commence
concert
cowbell
crackdown
cranky
crowfoot
crucial
crumpled
crusade
cubic
deadbolt
deckhand
dogsled
dosage
dragnet
drainage
dreadful
drifter
dropper
drumbeat
drunken
Dupont
dwelling
eating
edict
egghead
eightball
endorse
endow
enlist
erase
escape
exceed
eyeglass
eyetooth
facial
fallout
flagpole
flatfoot
flytrap
fracture
fragile
framework
freedom
frighten
gazelle
Geiger
Glasgow
glitter
glucose
goggles
goldfish
gremlin
guidance
hamlet
highchair
hockey
hotdog
indoors
indulge
inverse
involve
island
Janus
jawbone
keyboard
kickoff
kiwi
klaxon
lockup
merit
minnow
miser
Mohawk
mural
music
Neptune
newborn
nightbird
obtuse
offload
oilfield
optic
orca
payday
peachy
pheasant
physique
playhouse
Pluto
preclude
prefer
preshrunk
printer
profile
prowler
pupil
puppy
python
quadrant
quiver
quota
ragtime
ratchet
rebirth
reform
regain
reindeer
rematch
repay
retouch
revenge
reward
rhythm
ringbolt
robust
rocker
ruffled
sawdust
scallion
scenic
scorecard
Scotland
seabird
select
sentence
shadow
showgirl
skullcap
skydive
slingshot
slothful
slowdown
snapline
snapshot
snowcap
snowslide
solo
spaniel
spearhead
spellbind
spheroid
spigot
spindle
spoilage
spyglass
stagehand
stagnate
stairway
standard
stapler
steamship
stepchild
sterling
stockman
stopwatch
stormy
sugar
surmount
suspense
swelter
tactics
talon
tapeworm
tempest
tiger
tissue
tonic
tracker
transit
trauma
treadmill
Trojan
trouble
tumor
tunnel
tycoon
umpire
uncut
unearth
unwind
uproot
upset
upshot
vapor
village
virus
Vulcan
waffle
wallet
watchword
wayside
willow
woodlark
Zulu

Three Syllable Word List

adroitness
adviser
aggregate
alkali
almighty
amulet
amusement
antenna
applicant
Apollo
armistice
article
asteroid
Atlantic
atmosphere
autopsy
Babylon
backwater
barbecue
belowground
bifocals
bodyguard
borderline
bottomless
Bradbury
Brazilian
breakaway
Burlington
businessman
butterfat
Camelot
candidate
cannonball
Capricorn
caravan
caretaker
celebrate
cellulose
certify
chambermaid
Cherokee
Chicago
clergyman
coherence
combustion
commando
company
component
concurrent
confidence
conformist
congregate
consensus
consulting
corporate
corrosion
councilman
crossover
cumbersome
customer
Dakota
decadence
December
decimal
designing
detector
detergent
determine
dictator
dinosaur
direction
disable
disbelief
disruptive
distortion
divisive
document
embezzle
enchanting
enrollment
enterprise
equation
equipment
escapade
Eskimo
everyday
examine
existence
exodus
fascinate
filament
finicky
forever
fortitude
frequency
gadgetry
Galveston
getaway
glossary
gossamer
graduate
gravity
guitarist
hamburger
Hamilton
handiwork
hazardous
headwaters
hemisphere
hesitate
hideaway
holiness
hurricane
hydraulic
impartial
impetus
inception
indigo
inertia
infancy
inferno
informant
insincere
insurgent
integrate
intention
inventive
Istanbul
Jamaica
Jupiter
leprosy
letterhead
liberty
maritime
matchmaker
maverick
Medusa
megaton
microscope
microwave
midsummer
millionaire
miracle
misnomer
molasses
molecule
Montana
monument
mosquito
narrative
nebula
newsletter
Norwegian
October
Ohio
onlooker
opulent
Orlando
outfielder
Pacific
pandemic
Pandora
paperweight
paragon
paragraph
paramount
passenger
pedigree
Pegasus
penetrate
perceptive
performance
pharmacy
phonetic
photograph
pioneer
pocketful
politeness
positive
potato
processor
prophecy
provincial
proximate
puberty
publisher
pyramid
quantity
racketeer
rebellion
recipe
recover
repellent
replica
reproduce
resistor
responsive
retraction
retrieval
retrospect
revenue
revival
revolver
Sahara
sandalwood
sardonic
Saturday
savagery
scavenger
sensation
sociable
souvenir
specialist
speculate
stethoscope
stupendous
supportive
surrender
suspicious
sympathy
tambourine
telephone
therapist
tobacco
tolerance
tomorrow
torpedo
tradition
travesty
trombonist
truncated
typewriter
ultimate
undaunted
underfoot
unicorn
unify
universe
unravel
upcoming
vacancy
vagabond
versatile
vertigo
Virginia
visitor
vocalist
voyager
warranty
Waterloo
whimsical
Wichita
Wilmington
Wyoming
yesteryear
Yucatan