

certificate server so others can find it. Their own PGP software installs this key revocation certificate on their public keyrings and automatically prevents them from accidentally using your public key ever again. You can then generate a new private/public key pair and publish the new public key. You could send out one package containing both your new public key and the key revocation certificate for your old key.

What if you lose your private key?

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

But what can you do if you lose your private key, or if your private key is destroyed? You can't revoke it yourself, because you must use your own private key to revoke it, and you don't have it anymore. If you do not have a Designated Revoker for your key, someone specified in PGP who can revoke the key on your behalf, you must ask each person who signed your key to retire his or her certification. Then anyone attempting to use your key based on the trust of one of your introducers will know not to trust your public key.

For more information on Designated Revokers, see the *PGP Desktop Security User's Guide*.

Beware of snake oil

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

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

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

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

This is like selling automotive seat belts that look good and feel good, but snap open in 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 when you might not otherwise have done so if you had no cryptographic software at all. Perhaps you may never even discover that 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” that 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) and 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 don't know anything about the weaknesses of ECB mode. The very fact that they haven't even learned enough cryptography to know these elementary concepts is not reassuring. And they sometimes manage their DES keys in inappropriate or insecure ways. Also, these same software packages often include a second faster encryption algorithm that can be used instead of the slower DES. The author of the package often thinks his proprietary faster algorithm is as secure as DES, but after questioning him I usually discover that it's just a variation of my own brilliant scheme from college days. Or maybe he won't even reveal how his proprietary encryption scheme works, but assures me it's a brilliant scheme and I should trust it. I'm sure he believes that his algorithm is brilliant, but how can I know that without seeing it?

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

Even the really good software packages, that use DES in the correct modes of operation, still have problems. Standard DES uses a 56-bit key, which is too small by today's standards, and can 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 (<http://www.accessdata.com>) that sells a very low-cost package 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 \$2,000 to \$3,000, and used by the government for classified applications. It has strong cryptography, but requires some sort of special license from the government to buy this strong version. A commercial version of the STU-III is available that is watered down for NSA's convenience, and an export version is available that is even more severely weakened. Then there is the \$1,200 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 (nondigital) 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 OK.

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 made a lot of sense. I observed that practically no one in the commercial world of cryptography qualifies 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 United States 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 United States 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.

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

- 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 third principle method of suppression comes from the granting of 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 United States.

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

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

This document was created with Win2PDF available at <http://www.daneprairie.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.