CHAPTER SEVEN

The Evolution of PGP's Web of Trust

Phil Zimmermann
Jon Callas

When *Pretty Good Privacy* (PGP) first arrived in 1991, it was the first time ordinary people could use strong encryption that was previously available only to major governments.

PGP led to new opportunities for human rights organizations and other users concerned with privacy around the world, along with some oft-misunderstood legal issues that we'll touch on later.

One of the most influential aspects of PGP is its solution to the problem of connecting people who have never met and therefore never had a chance to exchange secure keys. This solution quickly earned the moniker "Web of Trust," which describes the way the system operates about as accurately as any phrase.

The trust mechanism in PGP has evolved a lot since the early releases. It's worth examining the reasons for the trust model and the way PGP has evolved to provide more robustness.

The Web of Trust also offers an interesting historical angle because it was an early peer-to-peer design, and arguably one of the first social networks.

Much has been written about PGP and practical public key cryptography, but to our dismay, we've found that much of what is written contains substantial inaccuracies. It is our goal in this chapter to describe the PGP trust model, as well as its implementation, standardization, and use. We also will put it in its historic and political context.

PGP and OpenPGP

PGP is software; *OpenPGP* is a standard and a protocol. PGP is also a registered trademark presently owned by PGP Corporation (as are the related terms "Pretty Good Privacy" and "Pretty Good").

Before PGP was a protocol, it was merely software. This is important because nowadays we use the term "PGP" to refer to the software made by PGP Corporation, which implements a number of standards and protocols, among them OpenPGP. Additionally, PGP is one implementation of many implementations of the OpenPGP standard. But much of the PGP story predates the OpenPGP standard, and there is no defined term to separate the pre-OpenPGP software from the pre-OpenPGP protocol.

The core OpenPGP protocol is defined in the IETF RFC 4880 and RFC 3156. The early PGP protocol is defined in RFC 1991. Consequently, we will use *PGP* to refer primarily to this early software, and *RFC 1991* to refer to the early protocol.

Trust, Validity, and Authority

Trust is a broad concept. It means many things in many contexts. In this discussion we will use a narrow, strict definition: the mechanism that is used to decide whether a key is valid. A key is *valid* if it is actually owned by the person who claims to own it. In other words, if the descriptive information traveling with the key is accurate, then the key is valid. You tell PGP whom you trust to introduce keys; in return, PGP tells you which keys are valid.

Thus, if a key says it is owned by the same person who owns the email address `"Alice" <alice@example.com>` and that information is correct, that key is valid (or accurate). If Bob believes the key is valid because Charlie signed that key, then Bob considers Alice's key valid because he trusts Charlie.*

Let us start with this definition:

> A key is *valid* if it is actually owned by the person who claims to own it.

We can then move on to define trust:

> *Trust* is the mechanism we use to decide that a key is valid.

Most people confuse trust and validity. Even those of us who know the Web of Trust best sometimes make mistakes. Validity is only a score and can be used only to determine whether the name on a key is accurate. Trust is a *relationship* that helps us determine validity.

You tell PGP whom you trust to sign keys, and in return PGP tells you which keys are valid. It does this by tallying up a score for each key depending on who signed the key and how much

* For a discourse both amusing and instructional on the use of these names, see the entry for John Gordon in "References" on page 129.

you trust the person who signed it. (PGP calls the signer an *introducer*, for reasons we'll explain later.)

For instance, if Bob believes that keys signed by Charlie are valid, Charlie can sign Alice's key and Bob will trust the key because he trusts Charlie.

A *trust model* is a general scheme that formalizes trust so a computer can automate the trust calculations. Let's look the basics of trust models.

Direct Trust

Direct trust is the most straightforward type of trust. In direct trust, Bob trusts that a certificate is Alice's because Alice gave it to him. It is the best trust model there is, and so simple that we described it without giving it a name. It is not only the simplest trust model, but at the end of the day it is the most, well, trustworthy!

People use direct trust all the time by doing things like putting an OpenPGP key fingerprint (which is itself a hash of an OpenPGP key) on their emails or business card. Even simpler, if I mail you my key, we're using direct trust. You trust that the key is valid because I am only hurting myself if that is wrong.

Often in a direct trust system, a certificate that holds the key is signed by that certificate itself. In other words, it is self-signed. Self-signing is useful because it provides a consistency check. We know that a self-signed certificate is either wholly accurate or wholly inaccurate.

The only problem with direct trust is that it doesn't scale very well to something the size of the Internet. That doesn't mean it's not useful; it just means it doesn't scale.

Hierarchical Trust

Hierarchical trust is also straightforward. In hierarchical trust, you trust that a certificate is valid because it was signed by someone whom you believe to be accurate, as in the example mentioned earlier. Bob considers Alice's certificate to be valid because Charlie signed it, and Bob trusts Charlie to be an authority on accurate signing.

There are a good number of companies that make it their business to be Charlies. GoDaddy and VeriSign, to name two, are widely trusted Certificate Authorities (CAs), and they are trusted because of the practices they follow in creating certificates. Part of these practices are that they have a relatively few number of keys that extend this trust. These keys are themselves held in *root certificates* (which are self-signed certificates). The key in this root certificate signs a key in another certificate. This can extend some number of times before we get to the actual certificate we're interested in.

To verify that certificate, we trace a chain of certificates. Alice's certificate is signed by a certificate that is signed by a certificate that is signed by the certificate that Jack built. Many

large corporations, governments, and so on have their own hierarchies that they create and maintain.

X.509 certificates of the sort that we use for SSL connections on the Web use hierarchical trust. If you go to Amazon.com, that web server has a certificate that was signed in a hierarchy that traces up to a root certificate that we trust. For these commercial Certificate Authorities, they typically use a depth of two or three.

Ah, now we hear you ask, "But how do we trust that root certificate?" The answer is: direct trust. Built into your web browser is a set of root certificates. You consider them valid because they are part of the software you installed. This is an important point: hierarchical trust ultimately derives from direct trust.

Hierarchical trust is straightforward, but has an obvious risk. If the authority makes a mistake, the effect of that mistake is great. In a notorious example of this problem, Microsoft uses a hierarchical trust system for its code-signing system, Authenticode. That hierarchy is maintained by VeriSign. A few years ago, some miscreants convinced VeriSign that they were Microsoft employees, but they were not. VeriSign issued Microsoft code-signing certificates to people who were not Microsoft. Fortunately, the scam was caught quickly. Had they gotten away with the improperly issued certificates, whoever ended up with those certificates would have been able to sign software with a Microsoft key, and from the system's viewpoint, that bogus software would have been Microsoft software.

Cumulative Trust

The drawback of hierarchical trust—that a hierarchy is brittle—leads us to the last major trust model, that of cumulative trust. Cumulative trust takes a number of factors into consideration and uses these factors to decide whether a certificate is valid.

Cumulative trust comes from an examination of the limitations of the hierarchy. In an idealized world, when everything goes right, hierarchical trust works best. We use hierarchical trust in the real world all the time. Passports, visas, driver's licenses, national identity systems, credit cards, and even an employer's credentials are all done through a hierarchical trust system.

However, the prime impulse of a security person is to look not at what can go right, but at what can go wrong. Hierarchies are brittle, and because they are brittle, there is a natural tendency to limit the scope of a hierarchy. In general, passports and driver's licenses are not unified, for example. Government documents rarely become credit documents. And because they are brittle, different hierarchies tend to look askance at each other. The badge that gets you into your workplace cannot get you onto a plane, despite a chain of trust! The badge comes from a hierarchy at your employer that is backed by identity documents into the banking and tax systems. At least in theory, they could be tied together. They are not tied, however, because of a lack of trust between the hierarchies.

The simplest cumulative trust system is represented by the question, "May I see two forms of ID, please?" The two forms of ID are a cumulative trust system. The security assumption is that it is much harder to have two erroneous trust paths than one. That idea, that the relying party collects at least one trust path, makes it cumulative. The accumulation might also use different types of trust. For example, when you give a merchant your credit card and the merchant asks to see your driver's license as well, that is also cumulative trust. Note that the credit card is a financial credential, and the driver's license is a quasi-identity credential. The merchant, however, accumulates the trust from the two credentials and completes the transaction.

Note that cumulative trust can encompass both direct trust and hierarchical trust. It is easy to set up in a cumulative trust system both direct trust and hierarchies of trust. We can also consider a hierarchy to be a special case of a cumulative system where we accumulate to one. Direct trust is also a special case of both a hierarchy and accumulation.

In public key infrastructure, the most widely used cumulative trust system is the PGP Web of Trust. However, cross-certification and bridge CAs are closely related to cumulative trust, if not precisely an accumulation system.

DEFINITIONS

Although most of the terms in this chapter are in common use in the security field, some definitions are useful, in order to keep us on the same page. PGP terminology is intentionally colloquial; it was created to be a common-language alternative to technical terms. Let us define some of those technical terms as well as the PGP-related terms:

PKI

: Public Key Infrastructure. A PKI is a set of technologies and mechanisms for creating, distributing, and maintaining public keys and certificates.

Certificate

: A certificate is a data structure that contains a public key, some information, and signatures that declare that the key and the information go together. Identity certificates are the most common. They bind a name and a public key (an email address is just a type of name) together. An attribute certificate binds information about the key holder to the key. For example, a certificate that says "the holder of this certificate is over 21 years of age" is an attribute certificate.

Certification

: A certification is a single binding of information and the key with a digital signature. X.509 certificates contain a single certification. OpenPGP certificates may contain a number of certifications.

  A certification (and thus a certificate) is created by a certificate authority (CA) when it creates the signature that binds the key and information together.

Key

OpenPGP's term for what other PKIs call a certificate. Whitfield Diffie coined this term because it is easier to say, type, and understand than "certificate." The average person has an intuitive sense of what a key is and what it is used for. Today, the term can be confusing in some contexts. An OpenPGP key (certificate) can contain many certifications, and in fact many related public keys, each governed by its own contexts and policies.

Keyring

A collection of keys. It may be a private database maintained by one user, but can also be shared among a number of users, as on the keyservers mentioned in the following item.

Keyserver

A directory that shares large collections of keys over a network. People can upload keys to a keyserver and use it as a reliable storage facility where they can keep all their key-related information up-to-date.

Sign

OpenPGP uses the colloquialism "sign" where other PKIs would say "certify." In OpenPGP's lingo, Alice signs (rather than certifies) Bob's key.

Introducer

A key that creates a certification. Thus, an introducer is OpenPGP's term for a certificate authority. Like the term *key*, Whitfield Diffie coined the term *introducer* as a more colloquial and human term than *certificate authority*.

The Basic PGP Web of Trust

For those who are familiar with other public key infrastructures (PKIs), PGP differs from the basic, hierarchical system with one tweak that everything else derives from. In the PGP trust model, all users are also certification authorities. Of course, not all authorities are created equal. Alice can certify Bob, but that doesn't mean Fran should accept the certification. This is why PGP uses the term *introducer* instead of *authority*. We have a natural tendency to trust an authority, but we attach realistic skepticism to an introduction.

"Phil, this is Bob," says Alice, making the introduction. Based on how accurate Phil thinks Alice is, he makes a decision. Phil can fully trust Alice as an introducer and accept Bob as valid. He can also wait until Jon says, "Hey, Phil, have you ever met Bob?" before he truly accepts the other person as accurately Bob.

There is a common misconception that since the Web of Trust is relative, in that all frames of reference are equally valid, there can be no recognized authorities and no organizational use. To think this is to confuse architecture and implementation. Many implementations and deployments hand a group a set of introductions to the end users, and they use them, the same way they accept the root certificates baked into other software.

But to understand the way it works, let us examine Figure 7-1, which shows the Web of Trust and all its facets. Each node in this figure is a key in a keyring. Each arrow represents a certification signature. An arrow going from Alice to Bob means that Alice has signed or certified Bob's key.

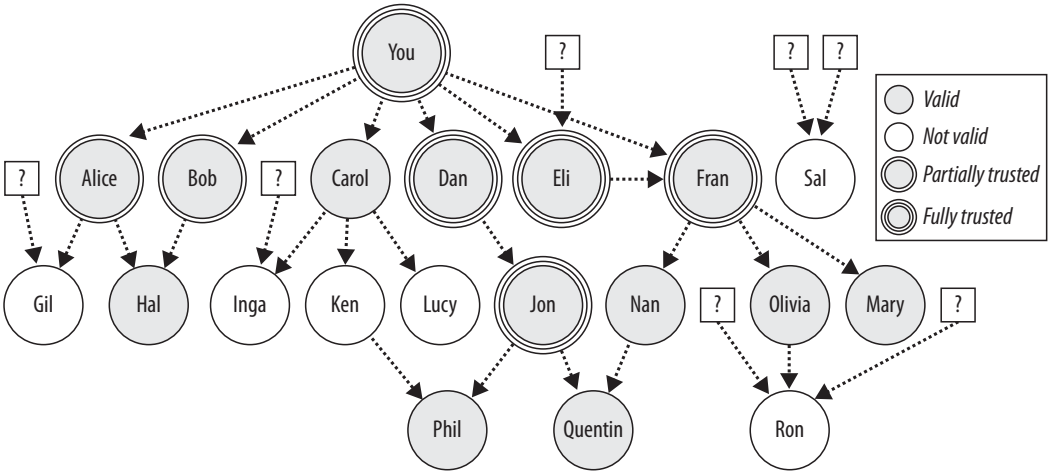

FIGURE 7-1. The PGP Web of Trust

At the top of Figure 7-1, the key labeled You is the owner's key. It is a triple circle to denote that it is fully trusted, which means that any key it signs will be valid. We also say that it is implicitly trusted, because you hold the private key as well as the public key. There are arrows that indicate that you have signed the keys belonging to Alice, Bob, Carol, Dan, Eli, and Fran.

There are also signatures that are dangling references to keys that are not in this keyring. For example, some key that you don't have has signed Eli's key.

The keys on the graph that are filled with gray are valid keys; we consider them to be accurate because the cumulative system of the Web of Trust calculates that they are valid. The accumulation works as follows: *partially trusted introducers* are denoted in the graph by a key with a double circle, and their signatures score one point; *fully trusted introducers* are denoted in the graph by a key with a triple circle, and their signatures score two points. If a key accumulates signatures from introducers that are worth two or more points, it is considered valid. Keys with zero or one point are not considered valid.

To see some examples, let's look at some specific subgraphs:

- Alice and Bob have each signed Hal. Therefore, Hal is valid, but Hal is not trusted.
- Gil is signed by Alice, but you only partially trust Alice, and so Gil's key is not valid.
- Carol has signed a number of keys that are also in this keyring, but none of them are valid. Interestingly, Ken signed Phil, which is valid, but not because Ken signed it.

- There is a tight cluster of keys with Dan, Eli, and Fran. You have signed each of them. Also, they are all fully trusted introducers. Eli's signature on Fran's key is redundant, because you directly certified it. Or perhaps your direct certification is redundant because Eli signed it.
- Nan, Olivia, and Mary are valid because Fran signed them. If we trace up from any of them, we see that there are two validity paths, one including Eli and the direct path from Fran to you.
- While you consider Jon valid because Dan signed his key, you also fully trust him. Consequently, Phil and Quentin are valid. Any other keys you discover that Jon signed are also going to be valid. This is of course true of the other fully trusted introducers, but Jon is someone you haven't directly trusted. The pair of Dan and Jon very closely resemble a hierarchical certification authority.
- Also note that while there are two upward paths from Quentin, one of them goes through Nan, who is valid but not trusted. Thus, while Nan has two upward paths of validity, Quentin only has one.
- Ron has three signatures, but none of them are from people you trust.
- Sal is signed by two dangling keys, which represent people unconnected to your Web of Trust.

We can now distinguish those two easy-to-confuse concepts of validity and trust another way: using the figure. Validity is a quality of a *node* (circle), whereas trust is a quality of the *edges* going between nodes. It is through the trust paths that we determine validity.

Rough Edges in the Original Web of Trust

The basic Web of Trust in early versions of PGP works very well as a cumulative trust system. However, there are a number of architectural and semantic rough edges in it. We fixed these rough edges in later versions of PGP, but we will review them here first.

Supervalidity

In Figure 7-1, Fran is a special key, in that she has a score of four: two from being signed by you and two from being signed by Eli. The Web of Trust makes no allowance for supervalid keys, yet intuitively there should be something special about Fran. There should also be something about a key that both Fran and Jon signed. We have yet to turn that intuition into a useful mechanism.

The social implications of signing keys

Despite this, many people are uncomfortable signing a key that belongs to someone they don't like.

Assigning trust can also be thorny from a social aspect. Still looking at Figure 7-1, suppose that you trust Carol with your very life (she might be your best friend), but you don't trust her to sign keys accurately, particularly because of her trusting nature. Similarly, Fran may be a picayune, compulsive nitpicker whom you can barely stand to be in the same room with, but you made her a fully trusted introducer for this very character flaw.

This underlines the point made when we defined trust at the beginning of this chapter: Web of Trust trust is a specialized trust limited to the sphere of validating keys, not a real-world trust.

Nonetheless, signing someone's key can be a very personal decision. Many people feel very strongly about it. Part of the strength of the Web of Trust is that this personal touch is part of PGP's zeitgeist. But it can also be a weakness that something so very simple—stating that you believe someone is who they claim to be—can become so emotionally charged. That's why the *xkcd* comic strip in Figure 7-2† is funny. For many people, certifying a key is an intensely personal thing.

FIGURE 7-2. Responsible behavior

A related emergent property of the Web of Trust is that key signatures acquire a cachet. They become like autographs, and develop social value.

Some of this social value is simply operational. If you have gone to the trouble to get several people you value to sign your key‡ and then get a new key, you have to obtain all of those signatures again. Many people keep their key longer than they should because the keys are signed by other people who are impressive in the right social circles, or whose relationships with the key holder are simply important emotionally.

This basic problem applies even to your own keys. There is presently no way to automatically roll forward the validity of your 1998 key to your 2008 key. If the old key signs the new key

† *http://imgs.xkcd.com/comics/responsible_behavior.png*, Randall Munroe, used by permission.

‡ Here the social value issue comes up again! Why should you get signatures from people you value rather than those who can make accurate statements?

and vice versa, this creates a manual roll-forward that is supported in the trust model, but it is not automated, and requires people to have both keys.

The social implications of the Web of Trust are its most fundamental and lasting rough edge. They create a social network out of security credentials. They make security credentials fun and personal, but by being fun and personal they acquire emotional value. We have no feelings about the SSL certificates on our website, but reviewing our PGP keys involves reviewing relationships.

PGP and Crypto History

A large number of myths unfortunately circulate through the technical community about PGP. It's important to know its true history to understand the technical choices made in the Web of Trust, so here we tell some of it.[§]

Early PGP

Phil began working on some of the early designs of PGP when he was a peace activist in the 1980s during the Nuclear Weapons Freeze campaign. The world was a different place then: Reagan was in the White House, Brezhnev was in the Kremlin, FEMA was telling cities to prepare evacuation plans, and millions of people feared the world was drifting inexorably toward Nuclear War. A million Americans marched for peace in Central Park.

It was in that political climate, in 1984, that Phil saw the need to develop what would later become PGP, both for protecting human rights overseas and for protecting grassroots political organizations at home. He started on the early design of PGP, but the more pressing matters of the peace movement postponed the bulk of the development effort until years later.

Phil wrote the first working version of PGP in 1991. He published PGP in the wake of Congressional discussion[‖] of requiring that all communications equipment and services have a "trap door" in them to permit government anti-criminal and counterterrorism activities.

While that discussion passed with no legislative outcome, Phil rushed to produce PGP 1.0 so that there would be freely available strong encryption in wide distribution while it was still legal to do so. One of PGP's explicit design goals was to metaphorically shoo the horses out of the barn before the door was closed.

§ The sources for PGP can be downloaded from *http://www.pgp.com/downloads/sourcecode*. This includes not only the sources to PGP Desktop, but the PGP Command Line sources and the PGP Universal GPL-modified sources. You can also find PGP's policies on assurance and special build requirements at *http://www.pgp.com/company/pgpassurance.html*.

‖ This was Senate Bill 266 of 1991. It never passed into law.

Although there were other message-encryption systems at the time, notably Privacy Enhanced Mail (PEM),[#] they were so tightly controlled that they were not suitable for mass deployment.[*]

PGP 1.0 captured the attention of many people, but it was flawed. In particular, it used the Bass-O-Matic cipher designed by Phil, in which Eli Biham found cryptographic flaws. So Phil—along with Hal Finney, Peter Gutmann, and Branko Lancaster—worked on its successor version, PGP 2.0.

PGP 2.0 replaced the Bass-O-Matic cipher with the International Data Encryption Algorithm (IDEA) cipher, designed by Xuejia Lai and James Massey. It also introduced the PGP trust model in a state that was close to its RFC 1991 form.

PGP 2 captured the imagination of many people who wanted to use strong cryptography in an ad hoc, unregulated environment. It became a grassroots phenomenon.

Patent and Export Problems

While cryptographically quite strong, PGP 2 had other issues, dealing with patents and export control. The IDEA cipher was patented, and the patent owner, Ascom-Tech AG, licensed it freely for noncommercial use but had inconsistent licensing for commercial users. The RSA public-key algorithm was patented in the United States by MIT, and was licensed to RSA Data Security Incorporated (RSADSI).

Phil met in 1986 with RSADSI's CEO, Jim Bidzos, and discussed licensing the RSA algorithm for freeware. Those meetings ended with Phil believing that he had permission to use the RSA algorithm in software so long as its cost was zero (i.e., that it was true freeware), and Bidzos believing the opposite. Years of disagreement about this agreement followed. Part of the difficulty in the legal status of PGP included the fact that the RSA algorithm was patented only in the United States, and that the actual owner of the RSA patent, MIT, was favorably disposed to PGP software.

The end result was that RSADSI created a library, RSAREF, for use in freeware and shareware, and PGP software was released in a version 2.5 that used RSAREF in the U.S. but broke compatibility with all previous versions. This meant that there were several versions of PGP:

- The pre-PGP 2.5 versions of the software, which were cut off from future development and support

[#] PEM was originally defined by John Linn and Steve Kent in RFC 1113, RFC 1114, and RFC 1115. These were revised with David Balenson and Burt Kaliski in RFC 1421, RFC 1422, RFC 1423, and RFC 1424.

[*] Jon Callas was at Digital Equipment Corporation at the time, and had a PEM certificate. Getting a PEM certificate involved a notary public and sending his passport to a certification authority by courier. He and his colleagues switched to PGP from PEM solely because it was impossible to get certificates to people who needed them in less than a month, or to people without passports.

- An "international" version of PGP software, developed completely outside of the United States (for export-control reasons), which used the original implementation of the RSA algorithm
- An "RSAREF" version of PGP software, developed inside the United States, using the RSAREF implementation of the RSA algorithm

In all of these versions, the IDEA cipher was free for noncommercial use, but not for commercial purposes.

While the intellectual-property disagreements were going on, Public Key Partners filed a complaint in 1992 with U.S. Customs, complaining that Phil was exporting cryptography without the appropriate licenses, namely the PGP program, which by then had spread throughout the Internet. That started the notorious investigation of Phil and PGP software. The investigation lasted until January 1996, when it was dropped.

There are many misconceptions about l'affaire Zimmermann that we can correct.

Phil was the target of a criminal investigation, but was not prosecuted. No criminal charges were filed about PGP against him or anyone else. Nor were there any lawsuits filed, nor any other legal action other than an investigation, nor did anyone spend any time in prison. The investigation covered the actual distribution of the PGP software itself, and not the PGP team's development practices. The PGP team consisted of developers in the United States, New Zealand, and Europe. The practice of developing software internationally was apparently never the subject of the investigation.

The U.S. government (in particular, the National Security Agency [NSA]) did not start the investigation in response to the software's existence. The popular misconception is that somehow the government did not like mass-distributed cryptography, and therefore started the investigation. U.S. Customs conducted the investigation in response to a complaint from Jim Bidzos. While the NSA was consulted in the matter, they apparently did not start it, and there is evidence that they were instrumental in dropping the investigation.†

The Crypto Wars

Part of the political context of the 1990s was what is often called *The Crypto Wars*. The Crypto Wars were the result of the changing role of cryptography in society. Until 1997, international regulation considered cryptography a weapon of war. Free and open cryptosystems were regulated as munitions in the United States, banned in France, and of mixed status throughout the rest of the world. In 1997, the U.S. reclassified cryptography as a dual-use item. France opened up nearly all cryptography in 1999, and export controls on cryptography were radically

† William Crowell, then deputy director of the NSA, stated in separate personal communications with Jon and Phil that he was consulted before the investigation started, and opined that an investigation was warranted, but that he also pressed to cancel the investigation, and claimed credit for its being canceled. Others within the Justice Department also later claimed credit for that decision.

liberalized in 2000. Those liberalizations have continued since then despite international concern about terrorism.

Part of the struggle to end the U.S. export controls on crypto involved the publication of PGP source code in its entirety, in printed book form. Printed books were and are exempt from the export controls. This happened first in 1995 with the publication of *PGP Source Code and Internals* (MIT Press). It happened again later when Pretty Good Privacy, Inc., published the source code of PGP in a more sophisticated set of books with specialized software tools that were optimized for easy optical character recognition (OCR) scanning of C source code. This made it easy to export unlimited quantities of cryptographic source code, rendering the export controls moot and undermining the political will to continue imposing the export controls.

Today, there has been nearly an about-face in government attitude about cryptography. National and international laws, regulations, and expectations about privacy, data governance, and corporate governance either imply or require the widespread use of strong cryptography. In 1990, the cultural attitude about cryptography could be described as, *Why do you need that? What do you have to hide?* Twenty years later, the cultural attitude is closer to, *Why don't you have it? Don't you understand that you have to protect your data?*

The definitive history of The Crypto Wars and the cultural shift in cryptography has not yet been written. Nonetheless, a good place to start is Steven Levy's *Crypto: How the Code Rebels Beat the Government Saving Privacy in the Digital Age* (Penguin).

From PGP 3 to OpenPGP

After the status of PGP 2 became calmer, Phil, along with Derek Atkins and Colin Plumb, started work on a new version of PGP software, PGP 3. PGP 3 contained a number of improvements to the RFC 1991 protocol, including:

- Support for multiple public-key pairs in a PGP key. In particular, its design called for separate signing and encryption keys, as a way to enforce key use.
- Support for DSA public-key signatures, as well as ElGamal for public-key encryption.
- Support for CAST5 and Triple DES for symmetric encryption.
- Replacing the MD5 hash function with SHA-1, after Hans Dobbertin found pseudo-collisions in its compression function (see "References" on page 129).

In 1996, Phil formed the company Pretty Good Privacy, Inc. with private investors and the company Viacrypt, which had produced commercial versions of PGP software. Since Viacrypt had released versions of the RFC 1991 system under the product name PGP 4, the PGP 3 cryptosystem was released as PGP 5.

At the time, members of the PGP software development team started to advocate a profile of the PGP 3 protocol they informally called Unencumbered PGP, a set of parameters that

eschewed algorithms encumbered with intellectual property. The team took Unencumbered PGP to the IETF as a successor to RFC 1991, and it became OpenPGP.

Enhancements to the Original Web of Trust Model

Although the model described earlier in this chapter is classic and fairly well known, few people know about the many extra features and enhanced sophistication that has been added to various later versions of PGP. As we recognized the limitations of the basic model, we added these elements to improve scaling and smooth over the rough edges.

The overarching areas of concern that came up again and again concerned revocation (or other reasons for keys becoming invalid), scaling problems, and the bloat caused as outdated signatures built up in keys. This section addresses each of those areas and a few other interesting enhancements.

Revocation

All PKIs need a way to revoke certificates. People are fallible creatures and they sometimes lose control of computers and keys. Systems can be lost or compromised, and the keys and certificates have to be declared invalid before they expire on their own.

Revocation is theoretically simple (although often still hard to propagate) in a hierarchical PKI such as X.509. Central authorities distribute revocations using the same channels they use to distribute the original authorizations to use keys.

The basic model for revocation

The original PGP Web of Trust described in the previous section offered two mechanisms for revocation:

Key revocation
: Someone who has lost control of her key must be able to revoke the whole thing; she can't depend on getting everyone who has signed it to revoke their signatures. This kind of revocation, like the previous one, is itself a kind of signature. (Signatures are the general way of transferring trusted information in PGP.) Signing a key with this key revocation signature invalidates all of its certifications.

Signature revocation
: A key can create a signature declaring that another signature is no longer valid. For instance, if Alice discovers that Bob cheated her into signing his key, she can revoke her certification of that key. If Charlie has also signed the key and doesn't revoke his signature, other people may trust Charlie and continue accepting Bob's key as valid. Note that this form of revocation is good for both data signatures and certification signatures.

Key revocation and expiration

Key revocation is a rough edge in all PKIs—perhaps the sharpest rough edge in public key cryptography. Although the PGP model is flexible and avoids some revocation pitfalls (such as the unmanageable sizes that certificate revocation lists tend to reach), it has its own issues with revocation. Here are some of them:

- There is no mechanism to distribute revocation information out-of-band, separately from the distribution of the keys themselves. If Alice revokes her key, she can place the revocation signature on a keyserver, but there is no way to broadcast this mechanism to everyone who has her key.
- There is no way for someone to revoke a lost key. It is not uncommon for a new PGP user to create a key, upload it to a keyserver, and then promptly forget the passphrase that protects it. The key has to remain there unused until it expires.
- Revocation (and expiration too, for that matter) includes no semantics for interpreting their meaning over time. For example, let us suppose that Alice creates a key in 2002 that expires in 2006. Suppose that she signs a document in 2003 and revokes her key in 2005. In 2008, what do we do with that signature? Certainly, in 2003, that signature was good, but is the signature still valid after she revokes her key?

  In practice, of course, it depends a lot on why she revoked her key. If she revoked it because she created a new one, the signature is still good. If she revoked it because identity thieves stole her laptop and wrote bad checks, then that signature should be considered a forgery, as there's no way for us to distinguish it from a forged, backdated signature.‡

These difficulties meant that keys were rarely revoked, except in obvious cases of key loss. At least in theory, it should be easy and desirable to revoke a key because it was superseded with a new key, or was otherwise retired. But because revocation was always looked upon with the worst-case scenario—that the key had truly been compromised—users rarely revoked keys.

A way to avoid revocation is to use expiration instead. The expiration date of a PGP key is held in a self-signature, which adds some additional flexibility because the key holder can change the expiration date simply by creating a new self-signature. Expiration, then, is not irrevocable. Unfortunately, people rarely used the expiration features.

Despite the improvements we describe next, revocation is still a conundrum.

Designated revokers

A designated revoker is a second key that has rights to generate revocation signatures for a primary key. A revocation coming from the designated revoker is as valid as one coming from the primary key itself.

‡ This particular problem could be helped by a third-party notary signature or timestamp signature. If the document in question has another signature from a notary or timestamping service, that extra signature states that Alice's signature in 2003 was valid at the time. In practice, however, no one ever does this.

Suppose that you designate Dan to be your designated revoker. When you do this, there is a self-signed signature in your key that states that Dan is your revoker. Additionally, this signature is itself marked as an irrevocable signature. The designation of a revoker must be irrevocable, because in the case of a true key compromise, the compromiser could otherwise just revoke the revoker.§

The designated revoker need not be an active key. You might, for example, create a key specifically as your revoker and store it offline in a safe, where it will remain until it is needed. The designated revoker feature provides a great deal of resilience against lost private keys, forgotten passphrases, and compromised keys. Everyone should define a designated revoker for their keys.

Designated revokers first appeared in PGP 3, and are a part of OpenPGP.

Freshness

Freshness is an alternative way to manage expiration and revocation. Freshness-based systems use standard expiration and revocation, but de-emphasize revocation over expiration. (See Rivest in "References" on page 129.)

Consider a key holder who creates a key that will expire in two weeks but re-creates the expiration signature every week. This holder has unilaterally constructed a freshness-based system. His need for revocation is minimized, since any given copy of his key expires in two weeks or less. In many cases, freshness can permit a holder to ignore revocation completely and rely on expiration.

Key signers can also use freshness-based systems to avoid revocation on their signatures.

Note, though, that freshness requires the signers to continually update their signatures. It also requires more rigorous policies in superseding replacement signatures.

Freshness-based OpenPGP systems were first described in a 2003 article by Jon, "Improving Message Security With a Self-Assembling PKI" (see "References" on page 129).

Reasons for revocation

As we noted earlier, there are many reasons to revoke a key, but only the most dramatic reason was covered by the initial implementation of the Web of Trust. Starting with OpenPGP, a revocation signature can be annotated to state whether the revocation is for key retiring, key compromise, or because the key has been superseded by a new key. This annotation can even include a human-readable string.

§ Irrevocable signatures were created at the same time as designated revokers because that feature created the need for them.

Scaling Issues

In its basic form, the Web of Trust scales excellently at the low end, up to a few thousand people. Inside a single organization, it even scales to tens or hundreds of thousands of people.

But large, disconnected networks of people may find it difficult to use the basic Web of Trust because there are few paths between people who do not already know each other. As the Web of Trust includes more nodes with relatively few edges, finding trust paths becomes difficult.

The Web of Trust works at its best with groups of people who have some connections. It does not work well with a large, ubiquitous network like the Internet. However, there are two saving graces—modern social networking reconstructs the sorts of small networks that are ideal for the Web of Trust. It may not work well on the Internet as a whole, but it works well in the Internet that most of us use.

Additionally, it is also important to remember the power of direct trust. If Alice has no connections to Zeke, she can always just ask him to send her his key.

Extended introducers

Extended introducers, also called *meta-introducers*, are a way to improve the scaling mechanisms of the Web of Trust by expanding it to multilevel hierarchies. The basic Web of Trust already supports multilevel hierarchies, but each signing node in the tree must be given trust. Extended introducers can automatically introduce introducers to a specified depth.

Examine Figure 7-1 again. Assume that Dan is given a meta-introducer signature with a depth of 1. With this trust signature, Jon's certificate is automatically a fully trusted introducer with no further signature. So is any other key that Dan signs.

Note that this gives Dan great power, the same as a root certificate authority with one level of delegation. In the real world, individuals are rarely given such power, but consider an organization broken up into departments: if a top-level key for the organization is given a meta-introducer signature and that key signs department keys, the departments automatically trust each other's keys without further interaction.

Extended introducers first appeared as part of OpenPGP.

Authoritative keys

Although extended introducers permit one key to be a trust point for a whole community, they are a limited solution to scaling that works only on a local level for a few relationships. The original Web of Trust succeeded in creating a usable PKI from nothing, when there were no authorities.

A much broader scaling mechanism is one of our most recent additions to the PGP Web of Trust. The notion of *authoritative keys*, first described by Jon in "Improving Message Security With a Self-Assembling PKI," helps build a fully distributed PKI.

The notion of authoritative keys is that there may be some certificates or keys that may be presumed to be genuine, not because they descend from cryptographic trust, but because they come from an appropriate authority, such as an Internet domain. For example, if we want to encrypt to a key identified by `alice@example.com`, we can accept the authority of `example.com` even if it has no trust anchors in common with us.

From a security standpoint, this isn't unreasonable—if `example.com` is being forged, then we all have many problems. Also, eventually DNSSEC will use digital signatures to protect the DNS domains, and this will make authoritative keys cryptographically secured. Authoritative keys rely on a network reality: although DNS may be easy to subvert on a small scale, it is hard to subvert on a large scale. Even the recent problems found in the DNS infrastructure have had the result of making DNS a reasonable trust point.

Authoritative keys address one of the most important issues of scaling: organizations have to be able to manage their own PKIs. The alternative is to have dozens or hundreds of hierarchies (which is in fact what happens with X.509 certificates). This alternative has its own scaling problems.

The notion of authoritative keys is also an integral component of the DKIM email-authentication protocol (see Allman et al. in "References" on page 129).

Signature Bloat and Harassment

PGP software systems typically agglomerate signatures on keys, which is the easiest way to handle the signatures and usually stays faithful to their meaning. However, there are three basic cases where this is not desired:

Replacement signatures
: If a new signature supersedes an older one, the older signature should be thrown away. For example, let's suppose Alice signs Bob's key in 2007 with an expiration date of 2008. Then, in 2008, after the first signature expires, she signs Bob's key again with an expiration date in 2009. It seems that the 2008 signature should replace the 2007 one. However, it is not always clear what makes a signature a successor to a previous one. It is also not clear in the basic Web of Trust whether the signature that was superseded should be deleted or kept as a historical record.

Expired signatures
: Consider the previous case, but let us suppose that after Alice's first signature expires, she had not yet created a new one. Should we delete the signature from Bob's key? An expired signature is not included in trust calculations, so why not delete it? The basic Web of Trust does not address this, and in general, implementations have taken the lazy way out and kept all expired signatures.

Unwanted signatures

Suppose Bob doesn't want Alice's signature on his key. There can be a number of reasons. Perhaps he has no idea who she is.[‖] Perhaps Bob despises Alice and wants no reminder of her. It is, after all, Bob's key.

There are two other related issues. In the Dudley/Snidely case, Dudley wants to sign Snidely's key for the public good, and Snidely understandably doesn't like this. So there's a conflict between Snidely's control of his own good and the public good at large. It is hard to come to some reconciliation of this conflict without burdening the concept of signing keys with the sort of value-based policies that we have tried so carefully to excise.

The second issue is that the basic Web of Trust provides no authentication of a signer. Let us suppose, for example, that the Dudley signature on Snidely's key comes not from Dudley but from Boris Badenov (a villain in another cartoon), who has created a false Dudley key strictly for the purpose of annoying Snidely.

We call this last case *signature harassment*. While we were developing OpenPGP, we discussed signature harassment internally, but never published any of these discussions, because harassment was so easy to carry out and so hard to guard against. The agglomerating policies of that generation of keyservers left them open to that form of signature abuse.

We instituted several new features over time to address these problems.

Exportable signatures

One of the main causes of signature bloat is the mechanics of importing keys. If Alice wants to import Phil's key and consider it valid, she is likely to need to sign it herself. If she accidentally (or intentionally) sends Phil's key back to a keyserver, it will have her signature on it.

A good way to stop this is to differentiate between signatures that someone makes as a public statement of certification and signatures they make for their own purposes, such as trust calculations. The former are *exportable*, whereas the latter are *nonexportable*.[#]

When a software implementation exports a key or sends it outside its own scope of control, it strips all nonexportable signatures. Additionally, when an implementation imports a key, it also strips nonexportable signatures.

This allows the Web of Trust to differentiate between certifications made for the world at large and certifications that exist for just the use of a single person to build a cryptographically secure set of valid keys.

Exportable signatures first appeared in PGP 3, and are a part of OpenPGP.

[‖] Both authors are irked by seeing signatures appear on our key that come from people we don't know.

[#] "Public" and "private" might have been better terms; "nonexportable" is a mouthful. But all the good terms were taken.

Key-editing policies

As another way to combat signature bloat, OpenPGP introduced two key-editing policies that the key owner can announce to a keyserver:

- Direct the server to permit only edits that have been authorized by the key owner
- Direct the server to replace an old copy of the key with a new copy, rather than simply agglomerate all key signatures

These two policies give a key owner control over the signatures on each of her keys. Alone, the first policy agglomerates signatures, but only under the owner's control. The two policies together give the key owner complete control over how her key is presented.

An addition to these policies is a URI embedded into the key that states where the definitive copy of the key resides. This location can be a keyserver or even a file accessed by HTTP or FTP. This URI permits a key owner to direct the world of users to a definitive copy of the key. The definitive copy helps the owner distribute revocation information as well. If the URI points to a keyserver, it also helps with signature revocations.

All of these policies, which first appeared as part of OpenPGP, are implemented as attributes of a self-signature, so a key can have different policies for each user ID on the key.

In-Certificate Preferences

While not strictly part of the trust model, an important feature introduced in OpenPGP allows it to store a number of user preferences in the self-signature of a key. For example, there are a number of options related to symmetric ciphers, compression functions, and hash functions.

OpenPGP permits users to state their preferences in an ordered list of values for each option that is placed in the user's self-signature.

This has a number of desirable features, some related to the Web of Trust. The option list is an authenticated store of a user's options and allows two implementations of OpenPGP to resolve differences of opinion between users.

For example, let's suppose Alice likes the colors red, blue, and purple, whereas Bob likes purple, green, white, and blue. In common, they share blue and purple, but in nearly opposite orders. The OpenPGP mechanism allows each of them to use whichever they like. Alice might use purple because it's Bob's preference, or blue because it's hers. What matters is that there is a guaranteed intersection. OpenPGP has a basic set of mandatory algorithms that ensure interoperability.

These in-certificate preferences are so useful that X.509 standards are in the process of adding their own in-certificate preferences.

The PGP Global Directory

The PGP Global Directory is a revised LDAP-based keyserver that improves upon previous generations of keyservers through better authentication and key editing. It works as follows:

- The Global Directory requires authentication for a key submitted to it, via an email round-trip. The Global Directory sends an email message with an authentication URL to all supplied email addresses. Only after the Global Directory is revisited via an authentication URL does it place the key in the directory. This procedure is similar to the email confirmations sent out by weblog, mailing list, and social network servers to prevent people from being signed up without their consent.
- The Global Directory permits the holder of an email address to delete a key for that email address via an email round-trip authentication, similar to adding a key.
- The Global Directory permits only one key per email address. This is not a perfect solution to key bloat, because there are many cases where a key holder could have more than one legitimate key for a specified email address. However, in this case, there is a secondary problem in how third parties decide which key to use. The oldest? The newest? Both? Listing only one key neatly sidesteps this issue, even though it doesn't address it to everyone's satisfaction.
- The Global Directory requires that a key be reverified semiannually. It sends each key an update email, and if there is no response to the email, the corresponding key is removed from the Global Directory.
- The Global Directory implements the OpenPGP key-editing policies described earlier.
- The Global Directory signs each key it holds with its own certification key. These signatures use freshness on a two-week interval. Thus, if a user deletes or revokes his key, it remains valid for two weeks at most in the Global Directory. This permits the Global Directory to forego its own revocation mechanisms.

These operational policies improve the accuracy of the Global Directory; its users know that every key on it was authenticated via an email exchange between the Directory and the key holder sometime in the last six months.

Note that some of these policies still have the potential for abuse. For example, Boris Badenov could still update Snidely Whiplash's key periodically. These issues are addressed through appropriate software that throttles requests, as well as anti-harassment mechanisms similar to those commonly used in anti-spam software.

The Global Directory is a consolidation of a number of ideas that circulated in the OpenPGP community, including:

- The Robot CA, first proposed by Phil Zimmermann, written about by Seth Schoen, and then refined and implemented by Kyle Hasselbacher (see "References" on page 129).
- The Self-Assembling PKI described earlier, designed by Jon Callas and Will Price

- Existing OpenPGP keyservers
- Email round-trip authentication, used by email mailing list servers such as Mailman (*http://www.list.org*), the GNU mailing list manager

Variable Trust Ratings

The cryptographer Ueli Maurer (see "References" on page 129) developed an extension to the PGP Web of Trust that allows for a continuous scale of trust assignments. In his model, the usual PGP scale would be zero, one-half, and one, with validity granted when enough trust accumulates to get to one. However, he permits any fraction to be assigned to an introducer. You can have one introducer given a value of 0.9 and another a value of 0.15.

Interesting Areas for Further Research

The Web of Trust has promoted human rights and international relations by allowing thousands of people to establish connections over the Internet without having to physically meet. It has also provided a fascinating environment for studying human-relationship-building and reputation. Several issues have come up over the years for which we don't currently have solutions; these can provide a starting point for another generation of research and coding.

Supervalidity

In Figure 7-1 earlier in this chapter, Fran is a special key, in that she has a score of four—two from your signature and two from Eli's. The Web of Trust makes no allowance for supervalid keys, yet intuitively one feels there should be something special about Fran. There should also be something special about a key that both Fran and Jon signed. We have yet to turn that intuition into a useful mechanism.

Social Networks and Traffic Analysis

Many people have observed that the Web of Trust forms a social network, a generalized directed graph that shows connections between people through their keys and signatures. This graph can be analyzed for social connection information.

If a given person subscribes to the theory that her signatures should be value-neutral, or even if she makes a point of signing "hostile" keys (such as Dudley signing Snidely's key), someone cannot assume a relationship between two people based upon the existence of a key signature. Moreover, PGP "key-signing parties," where a number of people get together and collectively certify each other's keys, blur the semantic meaning of the social network.

Neal McBurnett (see "References" on page 129) analyzed the network structure of the Web of Trust digraph. He examined the digraph for path lengths, connectedness, degree of scale, and other features.

Mark Reiter and Stuart Stubblebine created PATHSERVER (see "References" below), a way to evaluate multiple signature paths between keys.

These analyses are inspired by the Web of Trust and derive from the Web of Trust, but we must note that they are orthogonal to the Web of Trust proper. It is an integral feature of the Web of Trust that it consists of viewpoints; it may be considered relativistic, in that no frame of reference in the Web of Trust is inherently more valuable or trusted than any other. The *trust* portion of the Web of Trust relies completely on the user-specific trust markings and the weights that the key holder places on keys. The mesh of keys is an interesting object that we believe is useful on its own, and helps the overall use of the Web of Trust, but it is an orthogonal construct to the Web of Trust.

The Web of Trust's directed graph says something about the people in it. What it says, though, is open to both further research and debate.

References

Allman, E., J. Callas, M. Libbey, J. Fenton, and M. Thomas. *DomainKeys Identified Mail (DKIM) Signatures*, RFC 4871, *http://www.ietf.org/rfc/rfc4871.txt*.

Atkins, D., W. Stallings, and P. R. Zimmermann. *PGP Message Exchange Formats*, RFC 1991, *http://www.ietf.org/rfc/rfc1991.txt*.

Callas, J. "Improving Message Security With a Self-Assembling PKI," in *Proceedings of the 2nd Annual PKI Research Workshop*, Gaithersburg, MD, April 2003.

Callas, J., L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. *OpenPGP Message Format*, RFC 4880, *http://www.ietf.org/rfc/rfc4880.txt*.

Callas, J., L. Donnerhacke, H. Finney, and R. Thayer. *OpenPGP Message Format*, RFC 2440, *http://www.ietf.org/rfc/rfc2440.txt*.

Dobbertin, H. "Cryptanalysis of MD5 Compress," Announcement on the Internet, 1996.

Elkins, M., D. Del Torto, R. Levien, and T. Roessler. *MIME Security with OpenPGP*, *http://www.ietf.org/rfc/rfc3156.txt*.

Ellison, C. *SPKI Requirements*, RFC 2692, September 1999, *http://www.ietf.org/rfc/rfc2692.txt*.

Ellison, C., B. Frantz, B. Lampson, and R. Rivest. *SPKI Certificate Theory*, RFC 2693, September 1999, *http://www.ietf.org/rfc/rfc2693.txt*.

Garfinkel, S. *PGP: Pretty Good Privacy (http://oreilly.com/catalog/9781565920989/index.html)*. O'Reilly, 1995.

Gordon, J. "The Alice and Bob After Dinner Speech," given at the Zurich Seminar, April 1984, *http://downlode.org/etext/alicebob.html*.

Hasselbacher, K. "ROBOT CA," *http://www.toehold.com/robotca*.

Hasselbacher, K. "Robot CA: toward zero-UI crypto," *http://www.kuro5hin.org/story/2002/11/18/135727/66*.

Levy, S. *Crypto: How the Code Rebels Beat the Government—Saving Privacy in the Digital Age*, Diane Pub Co., 2003.

Maurer, U. "Modeling a Public-Key Infrastructure," in *Proceedings of the 1996 European Symposium on Research in Computer Security* (ESORICS '96), *Lecture Notes in Computer Science*, Springer-Verlag, Sept. 1996, Vol. 1146, pp. 325–350, *http://citeseer.ist.psu.edu/maurer96modelling.html*.

McBurnett, N. "PGP Web of Trust Statistics," *http://bcn.boulder.co.us/~neal/pgpstat*.

Reiter, M. and S. Stubblebine. "Path independence for authentication in large-scale systems," in *Proceedings of the 4th ACM Conference on Computer and Communications Security*, Zurich, Switzerland, April 1997, pp. 57–66, *http://stubblebine.com/97ccs.pdf*.

Reiter, M. and S. Stubblebine. "Resilient Authentication Using Path Independence," *IEEE Transactions on Computers*, Vol. 47, No. 12, December 1998.

R. Rivest. "Can We Eliminate Certificate Revocation Lists?" in *Proceedings of Financial Cryptography '98*, Springer Lecture Notes in Computer Science, No. 1465 (Rafael Hirschfeld, ed.), February 1998, pp. 178–183.

Schneier, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, Second Edition. John Wiley & Sons, 1996.

Schoen, S. "Casual PKI and making e-mail encryption easy," *http://www.advogato.org/article/391.html*.

Zimmermann, P. R. *The Official PGP User's Guide*. The MIT Press, 1995.

Zimmermann, P. R. *PGP: Source Code and Internals*. The MIT Press, 1997.