Usability of Security: A Case Study

Alma Whitten and J. D. Tygar
December 18, 1998
CMU-CS-98-155

Alma Whitten

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
alma@cs.cmu.edu

J. D. Tygar

EECS and SIMS
University of California
Berkeley, CA 94720
tygar@cs.berkeley.edu

Abstract

Human factors are perhaps the greatest current barrier to effective computer security. Most security mechanisms are simply too difficult and confusing for the average computer user to manage correctly. Designing security software that is usable enough to be effective is a specialized problem, and user interface design strategies that are appropriate for other types of software will not be sufficient to solve it.

In order to gain insight and better define this problem, we studied the usability of PGP 5.0, which is a public key encryption program mainly intended for email privacy and authentication. We chose PGP 5.0 because it has a good user interface by conventional standards, and we wanted to discover whether that was sufficient to enable non-programmers who know little about security to actually use it effectively. After performing both user testing and a cognitive walkthrough analysis, we conclude that PGP 5.0 is not sufficiently usable to provide effective security for most users.

In the course of our study, we developed general principles for evaluating the usability of computer security utilities and systems. This study is of interest not only because of the conclusions that we reach, but also because it can serve as an example of how to evaluate the usability of computer security software.

This publication was supported by Contract No. 102590-98-C-3513 from the United States Postal Service. The contents of this publication are solely the responsibility of the authors and do not necessarily represent the official views of the United States Postal Service.

Keywords: security, human-computer interaction, usability, public key cryptography, electronic mail, PGP.

1. Introduction

Security mechanisms are only effective when used correctly. Strong cryptography, provably correct protocols, and bug-free code will not provide security if the people who use the software forget to click on the encrypt button when they need privacy, give up on a communication protocol because they are too confused about which cryptographic keys they need to use, or accidentally configure their access control mechanisms to make their private data world-readable. Problems such as these are already quite serious: at least one researcher [1] has claimed that configuration errors are the probable cause of more than 90% of all computer security failures. Since average citizens are now increasingly encouraged to make use of networked computers for private transactions, the need to make security manageable for even untrained users has become critical [6].

As one remedy, we might attempt to educate the user population in the use of computer security. This is a worthy goal, but how is it to be accomplished? Employers may institute security training programs for their employees, but it is unlikely that more than a small fraction of home computer users will seek out classes on security management. News articles on the need for computer security probably make their readers more likely to opt for software that claims to be secure, but it seems doubtful that they have much educational effect beyond that.

We can try to minimize the need for security management by users, automating security mechanisms as much as possible, and invoking legal penalties for the trespassing that does occur. Again, these are worthwhile strategies, but insufficient as a solution. Automating the encryption of a communication channel can sometimes work well, since in the case of encryption it may be clear what the security mechanism is meant to accomplish. Automating the granting of access privileges to a Java applet is much less plausible, since only the user knows which local resources he or she considers important to keep private, and from whom. As for the legal penalties, how is the user to know when someone has trespassed, so that he or she can take legal action? The theft of private data does not necessarily leave any evidence behind. Even when there is evidence, a lawsuit is far more realistic as a course of action for a corporation than for an individual home computer user.

Finally, we can work on improving user interfaces for computer security, in the hope that we can find ways to make security sufficiently clear and intuitive that most people can use it effectively. It is tempting to think that this is a task that can be given to user interface designers after the security mechanisms have already been developed, but we believe that will not lead to effective security, for the following reasons:

1. Design techniques that create good user interfaces for other types of software are ill-fitted to creating user interfaces that enable effective security. In Section 2 we will discuss this claim in detail, and describe some specific properties that make the design of usable security software a difficult and specialized problem.

2. There is as yet no body of security-specific user interface design techniques. There are no recognized exemplars of good user interface design for security, and human-computer interaction (HCI) research to date has not focused much on security applications.

3. The development of security-specific user interface design techniques requires expertise in security as well as in HCI. Because security concepts are often subtle to understand, and because they must be used perfectly (see Section 2.1 below), an HCI expert who is unskilled in security is likely to produce a system where the security mechanisms are not used in exactly the correct fashion. In the case of other types of applications, getting the underlying concepts slightly wrong may still yield a system that could be fixed in the next release, but it is unacceptable in the unforgiving world of security. Moreover, as we argue below, good security software should teach a user about underlying security concepts – which certainly requires that its designer have a fundamental understanding of those concepts.

4. It is likely that many security mechanisms have been developed which simply cannot be made usable for the general population, as they are too complicated or too arcane. No user interface can make these complex mechanisms accessible. To avoid this, HCI considerations need to be addressed early in the process of designing the security mechanisms, not left until the finished design is handed off to a user interface team.

As a first step toward understanding what user interface design for effective security requires, we studied an existing security program, PGP 5.0[1] [2]. In Section 3 we give a brief description of PGP and discuss our reasons for choosing it as a case study. We found during the course of our study that the problem of how to evaluate the usability of security is also a difficult and specialized one; in Section 4 we describe the methods we considered and the requirements that we believe had to be taken into account in order to yield valid results. In Sections 5 and 6 we describe our direct usability analysis and our user testing, and in Section 7 we compare the results of both. We conclude with a brief discussion of related work. (We also include several appendices describing the experiment in greater detail.)

The problem of studying PGP 5.0 is of interest in its own right, but our fundamental motivation for this research was to give an example of how security and HCI professionals can attempt to analyze the usability of security. Security is a particularly tricky field and one that is unforgiving of mistakes, including the mistake of neglect. Errors are likely to be discovered only after security has been compromised, and possibly not even then. A good test of a security system needs to check not only that users can deploy security properly, but that they will deploy security properly. For example, one challenge is how to make sure that users deploy security mechanisms during the test, without coaching or prompting (which are unlikely to occur in real field situations).

We believe that it is possible to meaningfully check the usability of computer security, and our analysis is intended as a model for practitioners and researchers to use in future examinations of the usability of security software.

2. Defining usability for security

Why is usability for security a special problem? Why can't we just use established user interface techniques? We discuss five special characteristics of the usability problem for security and propose a working definition of usability that takes into account these properties.

2.1. Properties of the usability problem for security

- **The barn door property**

  The proverb about the futility of locking the barn door after the horse is gone is descriptive of an important property of computer security: once a secret has been left accidentally unprotected, even for a short time, there is no way to be sure that it has not already been read by an attacker. Because of this, user interface design for security needs to place a very high priority on making sure users understand their security well enough to keep from making potentially high-cost mistakes.

- **The weakest link property**

  It is well known that the security of a networked computer is only as strong as its weakest component. If a cracker can exploit a single error, the game is up. This means that users need to

[1] We chose to study the Apple Macintosh version of PGP 5.0, and the discussion in the rest of this paper refers specifically to that version. The Windows version is very similar, and the UNIX versions do not currently have a graphical user interface.

be guided to attend to all aspects of their security, not left to proceed through random exploration as they might with a word processor or a spreadsheet.

- **The unmotivated user property**

  Security is usually a secondary goal. People do not generally sit down at their computers wanting to manage their security; rather, they want to send email, browse web pages, or download software, and they want security in place to protect them while they do those things. It is easy for people to put off learning about security, or to optimistically assume that their security is working, while they focus on their primary goals. Designers of user interfaces for security should not assume that users will be motivated to read manuals or to go looking for security controls that are designed to be unobtrusive.

- **The abstraction property**

  Computer security management often involves security policies, which are systems of abstract rules for deciding whether to grant accesses to resources. The creation and management of such rules is an activity that programmers take for granted, but which may be alien and unintuitive to many members of the wider user population. User interface design for security will need to take this into account.

- **The lack of feedback property**

  The need to prevent dangerous errors makes it imperative to provide good feedback to the user, but providing good feedback for security management is a difficult problem. The state of a security configuration is usually complex, and attempts to summarize it are not adequate. Furthermore, the correct security configuration is the one which does what the user "really wants", and since only the user knows what that is, it is hard for security software to perform much useful error checking.

2.2. A working definition of usability for security

Taking those five properties into account, we propose that security software is usable if the people who are expected to use it

1. are reliably made aware of the security tasks they need to perform;
2. are able to figure out how to successfully perform those tasks;
3. don't make dangerous errors; and
4. are sufficiently comfortable with the interface to continue using it.

In our case study, we evaluate the usability of the encryption program PGP 5.0 [3, 8, 9] against this definition.

3. Case study: PGP

3.1. What PGP does

PGP, short for Pretty Good Privacy, is a program designed to provide the average person with access to "strong" cryptography, meaning cryptography that will be difficult or impossible to break even by the resources of a government [11]. It is widely available in both shareware and commercial versions, and is popular within the computer security community. Figure 1 shows the contents of the PGP 5.0 folder after installation.

PGP provides two main cryptographic operations for the user: encryption/decryption of files for privacy, and creation/verification of digital signatures of files for authentication. It is primarily intended for use with email, and the current Macintosh version includes a plug-in that allows it to be accessed directly from Eudora, a popular email program. It is also used for encrypting files for secure storage, and for digitally signing code binaries to provide evidence that they have not been modified during distribution.

PGP supports asymmetric (popularly known as "public key") cryptography[2], in which each user has a pair of keys: a public key, which must be made available to all the people the user wants to communicate securely with, and a corresponding private key, which is known only by the user and must be kept secret. Files encrypted with some person's public key can be decrypted only by that person's private key, so the user can send private email by acquiring the intended recipient's public key and using it to encrypt the message. Files signed with some person's private key can be verified as such by using that person's public key to check the signature; unless the signature was created using the corresponding private key, the signature check will fail.

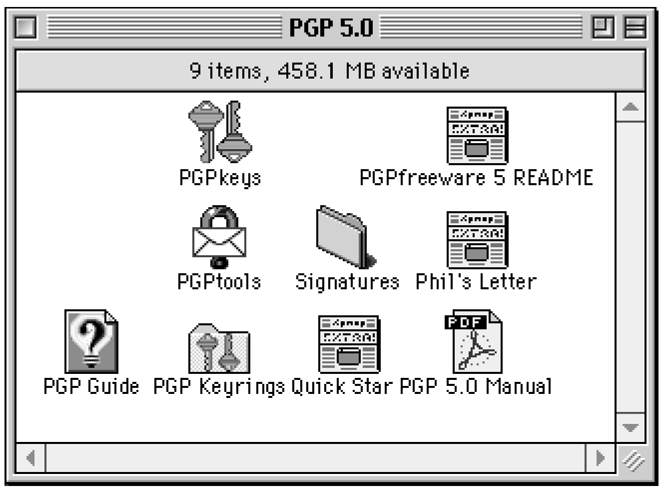


Figure 1

A great deal of key management is required in order to use PGP. At a minimum, the user must generate a key pair, give the public key to all the people with whom he or she wants to communicate securely, and get the public keys of all those people in return. These tasks are complicated by that fact that a crucial component of PGP's security is the accurate association of public keys with the real people who possess the matching private keys. When people who are personally known to each other meet to exchange public keys in person, this is not an issue, but often public keys are obtained through less reliable means, such as the key servers, which are generally accessible directories in which anyone may publish a key.

To help establish the association between public key and owner under those circumstances, people can digitally sign each other's keys, and publish their keys with signatures attached. Signing someone else's

[2] PGP also includes support for symmetric (popularly known as "private key") cryptography for file encryption, although we did not investigate this aspect of the program. In fact, we speculate that this feature, called "conventional encryption" by PGP, is confusing to users, since it uses a radically different key management structure and has different properties. As we see below grasping the concepts of asymmetric cryptography is difficult for people who are not security specialists – it is likely that needing to master not just one but two cryptographic structures will only exacerbate the confusion.

key is supposed to certify that the signer personally knows that key belongs to that person. A user who acquires someone's public key in an unreliable way may choose to trust that the key belongs to the right person if it has been signed by someone else whose key the user already has and trusts. The graph of keys connected by signatures that is created by this process is referred to as the "web of trust."

3.2. Why we chose PGP for our case study

As we mentioned in our introduction, it often tempting to think that security's usability problems can be solved just by adding nice graphical user interfaces where previously there were none. We believe that this is not the case, and we wanted to study a security application that did have a nice graphical user interface in order to gain an understanding of the kinds of usability problems that remained. PGP 5.0 is a recent program[3], with documentation [10] that claims: "The significantly improved graphical user interface makes complex mathematical cryptography accessible for novice computer users." We agree that PGP 5.0 has a nice user interface by the standards appropriate for a more traditional application such as a word processor or compression utility, so we considered it to be an ideal subject for our case study.

Another factor in our choice of PGP 5.0 was the fact that public key management is an important component of many security systems being proposed and developed today, and thus the problem of how to make such key management usable is one which is very generally applicable. PGP 5.0 is also a relatively small and specific security utility, so it is more amenable to a usability analysis than large, complex security systems.

3.3. Our usability standard for PGP

People who use email to communicate over the Internet have a need for security software that allows them to do so with privacy and authentication. The documentation and marketing literature for PGP presents it as a tool intended for that use by this large, diverse group of people, the majority of whom are not computer professionals. Referring back to our proposed definition of usability for security, we focused our case study on the following question.

> *If an average user of email feels the need for privacy and authentication, and acquires PGP with that purpose in mind, will PGP's current design allow that person to realize what needs to be done, figure out how to do it, and avoid dangerous errors, without becoming so frustrated that he or she decides to give up on using PGP after all?*

Restating the question in more detail, we want to know whether that person will, at minimum:

- understand that privacy is achieved by encryption, and figure out how to encrypt email and how to decrypt email received from other people;
- understand that authentication is achieved through digital signatures, and figure out how to sign email and how to verify signatures on email from other people;
- understand that in order to sign email and allow other people to send them encrypted email a key pair must be generated, and figure out how to do so;
- understand that in order to allow other people to verify their signature and to send them encrypted email, they must publish their public key, and figure out some way to do so;
- understand that in order to verify signatures on email from other people and send encrypted email to other people, they must acquire those people's public keys, and figure out some way to do so;
- manage to avoid such dangerous errors as accidentally failing to encrypt, trusting the wrong public keys, failing to back up their private keys, and forgetting their pass phrases; and

---

[3] At the time of this publication, PGP 6.0 has recently been released. The points raised in our case study may not apply to this newer version; however, this does not significantly diminish the value of PGP 5.0 as a subject for usability analysis.

- be able to succeed at all of the above within a few hours of reasonably motivated effort.

This is a minimal list of items that are essential to correct use of PGP. It does not include such important tasks as having other people sign the public key, signing other people's public keys, revoking the public key and publicizing the revocation, or evaluating the authenticity of a public key based on accompanying signatures and making use of PGP's built-in mechanisms for such evaluation.

4. Evaluation methods

We chose to evaluate PGP's usability through two methods: a cognitive walkthrough [14] in which we reviewed PGP's user interface directly and noted aspects of its design that failed to meet the usability standard described in Section 3.3; and a user test [12] performed in a laboratory with test participants selected to be reasonably representative of the general population of email users. The strengths and weaknesses inherent in each of the two methods made them useful in quite different ways, and it was more realistic for us to view them as complementary evaluation strategies [4] than to attempt to use the laboratory test to directly verify the points raised by the cognitive walkthrough.

Cognitive walkthrough is a usability evaluation technique modeled after the software engineering practice of code walkthroughs. To perform a cognitive walkthrough, the evaluators step through the use of the software as if they were novice users, attempting to mentally simulate what they think the novices' understanding of the software would be at each point, and looking for probable errors and areas of confusion. As an evaluation tool, cognitive walkthrough tends to focus on the learnability of the user interface (as opposed to, say, the efficiency), and as such it is an appropriate tool for evaluating the usability of security.

Although our analysis is most accurately described as a cognitive walkthough, it also incorporated aspects of another technique, heuristic evaluation [8]. In this technique, the user interface is evaluated against a specific list of high-priority usability principles; our list of principles is comprised by our definition of usability for security as given in Section 2.2 and its restatement specifically for PGP in Section 3.3. Heuristic evaluation is ideally performed by people who are "double experts," highly familiar with both the application domain and with usability techniques and requirements (including an understanding of the skills, mindset and background of the people who are expected to use the software). Our evaluation draws on our experience as security researchers and on additional background in training and tutoring novice computer users, as well as in theater, anthropology and psychology.

Some of the same properties that make the design of usable security a difficult and specialized problem also make testing the usability of security a challenging task. To conduct a user test, we must ask the participants to use the software to perform some task that will include the use of the security. If, however, we prompt them to perform a security task directly, when in real life they might have had no awareness of that task, then we have failed to test whether the software is designed well enough to give them that awareness when they need it. (The weakest link, unmotivated user, and lack of feedback properties apply here.) Furthermore, to test whether they are able to figure out how to use the security when they want it, we must make sure that the test scenario gives them some secret that they consider worth protecting, comparable to the value we expect them to place on their own secrets in the real world. Designing tests that take these requirements adequately into account is something that must be done carefully, and with the exception of some work on testing the effectiveness of warning labels [15], we have found little existing material on user testing that addresses similar concerns.

5. Cognitive walkthrough

5.1. Visual metaphors

The metaphor of *keys* is built into cryptologic terminology, and PGP's user interface relies heavily on graphical depictions of keys and locks. The PGPTools display, shown in Figure 2, offers four buttons to the user, representing four operations: Encrypt, Sign, Encrypt & Sign, and Decrypt/Verify, plus a fifth button for invoking the PGPKeys application. The graphical labels on these buttons indicate the encryption operation with an icon of a sealed envelope that has a metal loop on top to make it look like a closed padlock, and, for the decryption operation, an icon of an open envelope with a key inserted at the bottom. Even for a novice user, these are straightforward visual metaphors that help make the use of keys to encrypt and decrypt into an intuitive concept.

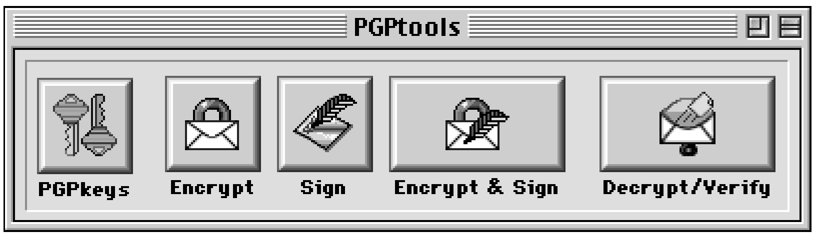


Figure 2

Still more helpful, however, would be an extension of the metaphor to distinguish between public keys for encryption and private keys for decryption; normal locks use the same key to lock and unlock, and the key metaphor will lead people to expect the same for encryption and decryption if it is not visually clarified in some way. Faulty intuition in this case may lead them to assume that they can always decrypt anything they have encrypted, an assumption which may have upsetting consequences. Different icons for public and private keys, perhaps drawn to indicate that they fit together like puzzle pieces, might be an improvement.

Signatures are another metaphor built into cryptologic terminology, but the icon of the blue quill pen that is used to indicate signing is problematic. People who are not familiar with cryptography probably know that quills are used for signing, and will recognize that the picture indicates the signature operation, but what they also need to understand is that they are using their private keys to generate signatures. The quill pen icon, which has nothing key-like about it, will not help them understand this and may even lead them to think that, along with the key objects that they use to encrypt, they also have quill pen objects that they use to sign. Quill pen icons encountered elsewhere in the program may be taken to be those objects, rather than the signatures that they are actually intended to represent. A better icon design might keep the quill pen to represent signing, but modify it to show a private key as the nib of the pen, and use some entirely different icon for signatures, perhaps something that looks more like a bit of inked handwriting and incorporates a keyhole shape.

Signature verification is not represented visually, which is a shame since it would be easy for people to overlook it altogether. The single button for Decrypt/Verify, labeled with an icon that only evokes decryption, could easily lead people to think that "verify" just means "verify that the decryption occurred correctly." Perhaps an icon that showed a private key unlocking the envelope and a public key unlocking the signature inside could suggest a much more accurate model to the user, while still remaining simple enough to serve as a button label.

5.2. Different key types

Originally, PGP used the popular RSA algorithm for encryption and signing. PGP 5.0 uses the Diffie-Hellman/DSS algorithms. The RSA and Diffie-Hellman/DSS algorithms use correspondingly different types of keys. The makers of PGP would prefer to see all the users of their software switch to use of Diffie-Hellman/DSS, but have designed PGP 5.0 to be backward compatible and handle existing RSA keys when necessary. The lack of forward compatibility, however, can be a problem: if a file is encrypted for several recipients, some of whom have RSA keys and some of whom have Diffie-Hellman/DSS keys, the recipients who have RSA keys will not be able to decrypt it unless they have upgraded to PGP 5.0; similarly, those recipients will not be able to verify signatures created with Diffie-Hellman/DSS without a software upgrade.

PGP 5.0 alerts its users to this compatibility issue in two ways. First, it uses different icons to depict the different key types: a blue key with an old fashioned shape for RSA keys, and a brass key with a more modern shape for Diffie-Hellman/DSS keys, as shown in Figure 3. Second, when users attempt to encrypt documents using mixed key types, a warning message is displayed to tell them that recipients who have earlier versions of PGP may not be able to decrypt it.

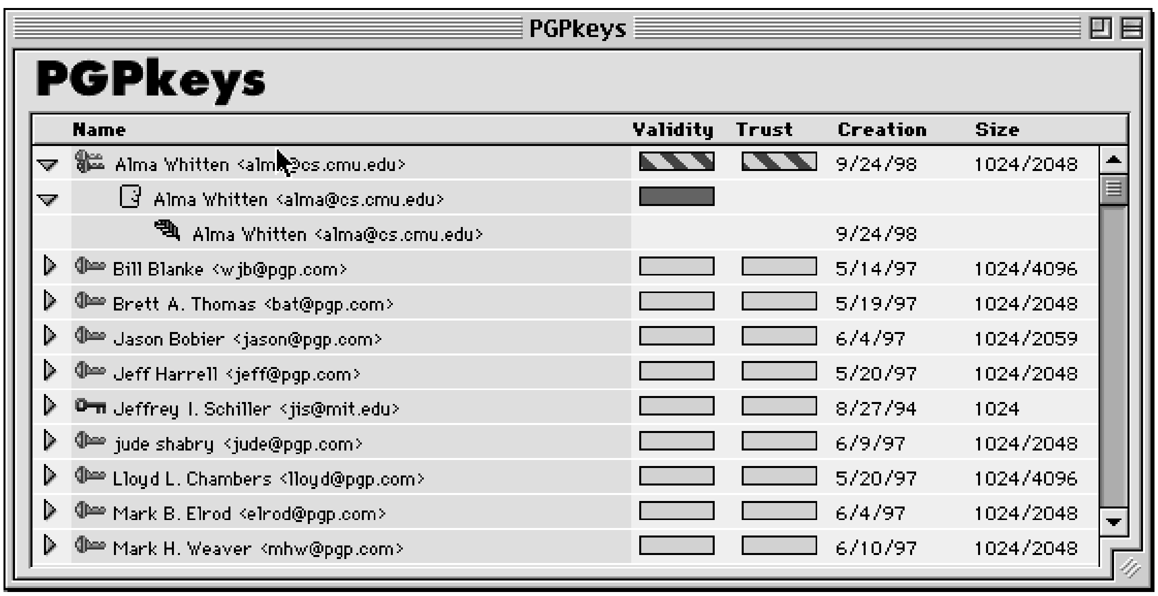


Figure 3

Unfortunately, information about the meaning of the blue and brass key icons is difficult to find, requiring users either to go looking through the 132 page manual, or to figure it out based on the presence of other key type data. Furthermore, other than the warning message encountered during encryption, explanation of why the different key types are significant (in particular, the risk of forward compatibility problems) is given only in the manual. Double-clicking on a key pops up a Key Properties window, which would be a good place to provide a short message about the meaning of the blue or brass key icon and the significance of the corresponding key type.

It is most important for the user to pay attention to the key types when choosing a key for message encryption, since that is when mixed key types can cause compatibility problems. However, PGP's dialog box (see Figure 4) presents the user with the metaphor of choosing people (recipients) to receive the message, rather than keys to encrypt the message with. This is not a good design choice, not only because the human head icons obscure the key type information, but also because people may have multiple keys, and it is counterintuitive for the dialog to display multiple versions of a person rather than the multiple keys that person owns.

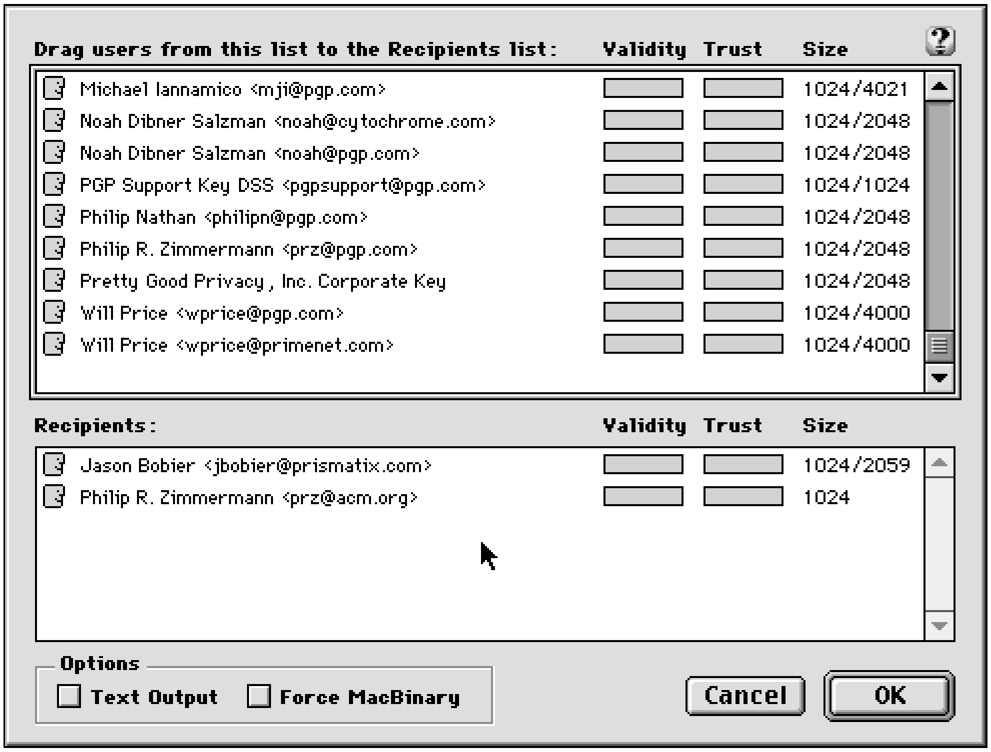


Figure 4

5.3. Key server

Key servers are publicly accessible (via the Internet) databases in which anyone can publish a public key joined to a name. PGP is set to access a key server at MIT by default, but there are others available, most of which are kept up to date as mirrors of each other. PGP offers three key server operations to the user under the Keys pull-down menu in PGPKeys: Get Selected Key, Send Selected Key, and Find New Keys. The first two of those simply connect to the key server and perform the operation. The third asks the user to type in a name or email address to search for, connects to the key server and performs the search, and then tells the user how many keys were returned as a result, asking whether or not to add them to the user's key ring.

The first problem we find with this presentation of the key server is that users may not realize it exists, since there is no representation of it in the top level of the PGPKeys display. Putting the key server operations under a Key Server pull-down menu would be a better design choice, especially since it is worthwhile to encourage the user to make a mental distinction between operations that access remote machines and those that are purely local. We also think that it should be made clearer that a remote machine is being accessed, and that the identity of the remote machine should be displayed. Often the "connecting…receiving data…closing connection" series of status messages that PGP displayed flashed by almost too quickly to be read.

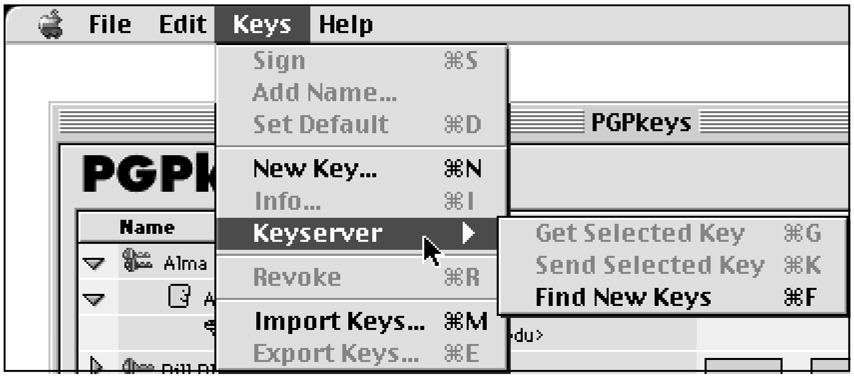


Figure 5

At present, PGPKeys keeps no records of key server accesses. There is nothing to show whether a key has been sent to a key server, or when a key was fetched or last updated, and from which key server the key was fetched or updated. This is information that might be useful for user key management, and also to verify to the user that key server operations did complete successfully. Adding this record keeping to the information displayed in the Key Properties window would improve PGP.

Key revocation, in which a certificate is published to announce that a previously published public key should no longer be considered valid, generally implies the use of the key server to publicize the revocation. PGP's key revocation operation does not send the resulting revocation certificate to the key server, which is probably as it should be, but there is a risk that some users will assume that it does do so, and fail to take that action themselves. A warning that the created revocation certificate has not yet been publicized would be appropriate.

5.4. Key management policy

PGP maintains two ratings for each public key in a PGP key ring. These ratings may be assigned by the user or derived automatically. The first of these ratings is *validity* which is meant to indicate how sure the user is that the key is safe to encrypt with (i.e., that it does belong to the person whose name it is labeled with). A key may be labeled as completely valid, marginally valid, or invalid. Keys that the user generates are always completely valid. The second of these ratings is *trust* which indicates how much faith the user has in the key (and implicitly, the owner of the key) as a certifier of other keys. Similarly, a key may be labeled as completely trusted, marginally trusted, or untrusted, and the user's own keys are always completely trusted.

What the user may not realize, unless they read the manual very carefully, is that there is a policy built into PGP that automatically sets the validity rating of a key based on whether it has been signed by a certain number of sufficiently trusted keys. This is dangerous. There is nothing to prevent users from innocently assigning their own interpretations to those ratings and setting them accordingly (especially since "validity" and "trust" have different colloquial meanings), and it is certainly possible that some people might make mental use of the validity rating while disregarding and perhaps incautiously modifying the trust ratings. PGP's ability to automatically derive validity ratings can be useful, but the fact that PGP is doing so needs to be made obvious to the user.

5.5. Irreversible actions

Some user errors are reversible, even if they require some time and effort to reconstruct the desired state. The ones we list below, however, are not, and potentially have unpleasant consequences for the user, who might lose valuable data.

- **Accidentally deleting the private key**

  A public key, if deleted, can usually be gotten again from a key server or from its owner. A private key, if deleted and not backed up somewhere, is gone for good, and anything encrypted with its corresponding public key will never be decryptable, nor will the user ever be able to make a revocation certificate for that public key. PGP responds to any attempt to delete a key with the question "Do you really want to delete these items?" This is fine for a public key, but attempts to delete a private key should be met with a warning about the possible consequences.

- **Accidentally publicizing a key**

  Information can only be added to a key server, not removed. A user who is experimenting with PGP may end up generating a number of key pairs that are permanently added to the key server, without realizing that these are permanent entries. It is true that the effect of this can be partially addressed by revoking the keys later (or waiting for them to expire), but this is not a satisfactory solution. First, even if a key is revoked or expired, it remains on the key server. Second, the notions of revocation and expiration are relatively sophisticated concepts; concepts that are likely to be unfamiliar to a novice user. For example, as discussed in the above bullet, the user may accidentally lose the ability to generate a revocation certificate for a key. This is particularly likely for a user who was experimenting with PGP and generating a variety of test keys that they intend to delete. One way to address this problem would be to warn the user when he sends a key to a server that the information being sent will be a permanent addition.

- **Accidentally revoking a key**

  Once the user revokes a public key, the only way to undo the revocation is to restore the key ring from a backup copy. PGP's warning message for the revocation operation asks "Are you sure you want to revoke this key? Once distributed, others will be unable to encrypt data to this key." This message doesn't warn the user that, even if no distribution has taken place, a previous backup of the key ring will be needed if the user wants to undo the revocation. Also, it may contribute to the misconception that revoking the key automatically distributes the revocation.

- **Forgetting the pass phrase**

  PGP suggests that the user make a backup revocation certificate, so that if the pass phrase is lost, at least the user can still use that certificate to revoke the public key. We agree that this is a useful thing to do, but we also believe that only expert users of PGP will understand what this means and how to go about doing so (under PGP's current design, this requires the user to create a backup of the key ring, revoke the public key, create another backup of the key ring that has the revoked key, and then restore the key ring from the original backup).

- **Failing to back up the key rings**

  We see two problems with the way the mechanism for backing up the key rings is presented. First, the user is not reminded to back up the key rings until he or she exits PGPKeys; it would be better to remind as soon as keys are generated, so as not to risk losing them to a system crash. Second, although the reminder message tells the user that it is important to back up the keys to some medium other than the main hard drive, the dialog box for backing up presents the main

PGP folder as a default backup location. Since most users will just click the "Okay" button and accept the default, this is not a good design.

5.6. Consistency

When PGP is in the process of encrypting or signing a file, it presents the user with a status message that says it is currently "encoding." It would be better to say "encrypting" or "signing", since seeing terms that explicitly match the operations being performed helps to create a clear mental model for the user, and introducing a third term may confuse the user into thinking there is a third operation taking place. We recognize that the use of the term "encoding" here may simply be a programming error and not a design choice per se, but we think this is something that should be caught by usability-oriented product testing.

5.7. Too much information

In previous implementations of PGP, the supporting functions for key management (creating key rings, collecting other people's keys, constructing a "web of trust") tended to overshadow PGP's simpler primary functions, signing and encryption. PGP 5.0 separates these functions into two applications: PGPKeys for key management, and PGPTools for signing and encryption. This cleans up what was previously a rather jumbled collection of primary and supporting functions, and gives the user a nice simple interface to the primary functions. We believe, however, that the PGPKeys application still presents the user with far too much information to make sense of, and that it needs to do a better job of distinguishing between basic, intermediate, and advanced levels of key management activity so as not to overwhelm its users.

Currently, the PGPKeys display (see Figure 3) always shows the following information for each key on the user's key ring: owner's name, validity, trust level, creation date, and size. The key type is also indicated by the choice of icon, and the user can toggle the display of the signatures on each key. This is a lot of information, and there is nothing to help the user figure out which parts of the display are the most important to pay attention to. We think that this will cause users to fail to recognize data that is immediately relevant, such as the key type; that it will increase the chances that they will assign wrong interpretations to some of the data, such as trust and validity; and that it will add to making users feel overwhelmed and uncertain that they are managing their security successfully.

We believe that, realistically, the vast majority of PGP's users will be moving from sending all of their email in plain text to using simple encryption when they email something sensitive, and that they will be inclined to trust all the keys they acquire, because they are looking for protection against eavesdroppers and not against the sort of attack that would try to trick them into using false keys. A better design of PGPKeys would have an initial display configuration that concentrated on giving the user the correct model of the relationship between public and private keys, the significance of key types, and a clear understanding of the functions for acquiring and distributing keys. Removing the validity, trust level, creation date and size from the display would free up screen area for this, and would help the user focus on understanding the basic model well. Many security experts may feel that the trust and validity information for keys is of importance, but as our experimental results establish, the PGP model is simply too large to be swallowed by users in one bite. We must simplify, and we cannot cut the core functions of the system.

A smaller set of more experienced users will probably care more about the trustworthiness of their keys; perhaps these users do have reason to believe that the contents of their email is valuable enough to be the target of a more sophisticated, planned attack, or perhaps they really do need to authenticate a digital signature as coming from a known real world entity. These users will need the information given by the signatures on each key. They may find the validity and trust labels useful for recording their assessments of those signatures, or they may prefer to glance at the actual signatures each time. It would be worthwhile to allow users to add the validity and trust labels to the display if they want to, and to provide easily accessible help for users who are transitioning to this more sophisticated level of use. But this would only make sense if the automatic derivation of validity by PGP's built-in policy were turned off for these users, as discussed in Section 5.4.

Key size is really only relevant to those who actually fear a cryptographic attack, and could certainly be left as information for the Key Properties dialog, as could the creation date. Users who are sophisticated enough to make intelligent use of that information are certainly sophisticated enough to go looking for it.

5.8. Documentation

After installation, the PGP 5.0 folder contains five items that appear to be documentation: three of these items, labeled "Phil's Letter," "PGPFreeware 5 README," and "QuickStart" are displayed as identical newspaper page icons; another, labeled "PGP Guide," has the Macintosh Help (question mark in light bulb) icon; and the fifth is an Adobe PDF document icon labeled "PGP 5.0 Manual."

We believe that PGP's documentation should be optimized for two purposes. It should provide an easily locatable short initial guide to help the user get a basic familiarity with the program, and it should provide a larger, comprehensive reference that allows the user to quickly and directly access information relevant to specific questions. The better the design of the program interface, the less need there should be for that short initial guide. Ideally it should serve as a backup for when the accessibility and clarity of the interface itself is not enough.

"PGPFreeware 5 README" and "QuickStart" both look like they might be intended to serve as that short initial guide, but neither is particularly useful for that purpose. The former is a brief set of comments related to fixing problems that might arise with the functioning of the program, and the latter is a collection of advice about various parts of PGP's functionality, too terse to be of much use to non-experts. "Phil's Letter" is basically a chatty advertisement for PGP and the philosophy behind it. Trying to access "PGP Guide" results in the message "This is a guide file for use with Apple Guide. It can only be run from within the application it is associated with." (This is a MacOS issue.)

The content from "PGP Guide" is available as the PGP Help that can be accessed under the Apple Help menu when PGPTools or PGPKeys is running (but PGP Help is not available if the user is working through PGPMenu). PGP Help is arranged into a top-down collection of informational snippets that in presentation format look like they should be used for direct access in answer to specific questions, but it really has the content of a short initial guide. We think it would be more useful to present this content in a linear format, so that users can recognize it as the short initial guide and not as a shallow, limited form of the more comprehensive reference.

As a resource for finding answers to specific questions, PGP Help is quite limited. First, while using the PGP applications, there is no way to go directly from an application context to help related to that context; if the user is confused while performing a particular task they have no better option than to manually search through PGP Help for topic headings that appear relevant. Second, the information accessible through PGP Help is at a very basic level, and, with a few exceptions, there is no way to ask PGP Help for more detailed information on a particular point.

The more comprehensive reference is certainly the "PGP 5.0 Manual" which is a 132 page Adobe PDF document that can either be printed out or displayed using the free Adobe Acrobat Reader. Much of the information in the manual could be very useful as context sensitive on-line help. However, the PDF format is not a great choice for repeated on-line access; HTML would be quicker to display and easier to search. Printing out the manual is fine for those who work at a desktop computer, but laptop users will not want to carry a large printout around.

6. User test

6.1. Purpose

Our user test was designed to determine how close PGP 5.0 comes to the usability standard we proposed for it in section 3.3. Furthermore, we wished to test this in as general a manner as possible. We gave our participants a test scenario that was both plausible and appropriately motivating, and then avoided interfering with their attempts to carry out the security tasks that we gave them. We felt that the information we would gain from such a general test was of greater value than what we would get from a test designed to specifically and carefully verify the points we raised in the direct analysis. (The latter sort of test would also be valuable, but time and resource limitations forced us to prioritize).

6.2. Description

6.2.1. Test design

Our test scenario was that the participant had volunteered to help with a political campaign and had been given the job of campaign coordinator (the party affiliation and campaign issues were left to the participant's imagination, so as not to offend anyone). The participant's task was to send out campaign plan updates to the other members of the campaign team by email, using PGP for privacy and authentication. Since presumably volunteering for a political campaign implies a personal investment in the campaign's success, we hoped that the participants would be appropriately motivated to protect the secrecy of their messages. Appendix E contains the document used to brief the participants on the test scenario.

Since PGP does not handle email itself, it was necessary to provide the participants with an email handling program to use. We chose to give them Eudora, since that would allow us to also evaluate the success of the Eudora plug-in that is included with PGP. Since we were not interested in testing the usability of Eudora (aside from the PGP plug-in), we gave the participants a brief Eudora tutorial before starting the test, and intervened with assistance during the test if a participant got stuck on something that had nothing to do with PGP.

After briefing the participants on the test scenario and tutoring them on the use of Eudora, they were given an initial task description (attached as Appendix F) which provided them with a secret message (a proposed itinerary for the candidate), the names and email addresses of the campaign manager and four other campaign team members, and a request to please send the secret message to the five team members in a signed and encrypted email. In order to complete this task, a participant had to generate a key pair, get the team members' public keys, make their own public key available to the team members, type the (short) secret message into an email, sign the email using their private key, encrypt the email using the five team members' public keys, and send the result. In addition, we designed the test so that one of the team members had an RSA key while the others all had Diffie-Hellman/DSS keys, so that if a participant encrypted one copy of the message for all five team members (which was the expected interpretation of the task), they would encounter the mixed key types warning message. Participants were told that after accomplishing that initial task, they should wait to receive email from the campaign team members and follow any instructions they gave.

Each of the five campaign team members was represented by a dummy email account and a key pair which were accessible to the test monitor through a networked laptop. The campaign manager's private key was used to sign each of the team members' public keys, including her own, and all five of the signed public keys were placed on the default key server at MIT, so that they could be retrieved by participant requests. Under certain circumstances, the test monitor posed as a member of the campaign team and sent email to the participant from the appropriate dummy account. These circumstances were:

1. The participant sent email to that team member asking a question about how to do something. In that case, the test monitor sent the minimally informative reply consistent with the test scenario, i.e. the minimal answer that wouldn't make that team member seem hostile or ignorant beyond the bounds of plausibility[4].

2. The participant sent the secret in a plaintext email. The test monitor then sent email posing as the campaign manager, telling the participant what happened, stressing the importance of using encryption to protect the secrets, and asking the participant to try sending an encrypted test email before going any further. If the participant succeeded in doing so, the test monitor (posing as the campaign manager) then sent an updated secret to the participant in encrypted email and the test proceeded as from the beginning.

3. The participant sent email encrypted with the wrong key. The test monitor then sent email posing as one of the team members who had received the email, telling the participant that the team member was unable to encrypt the email and asking whether the participant had used that team member's key to encrypt.

4. The participant sent email to a team member asking for that team member's key. The test monitor then posed as that team member and sent the requested key in email.

5. The participant succeeded in carrying out the initial task. They were then sent a signed, encrypted email from the test monitor, posing as the campaign manager, with a change for the secret message, in order to test whether they could decrypt and read it successfully. If at that point they had not done so on their own, they received email prompting to remember to back up their key rings and to make a backup revocation certificate, to see if they were able to perform those tasks. If they had not sent a separately encrypted version of the message to the team member with the RSA key, they also received email from the test monitor posing as that team member and complaining that he couldn't decrypt the email message.

6. The participant sent email telling the team member with the RSA key that he should generate a new key or should upgrade his copy of PGP. In that case the test monitor continued sending email as that team member, saying that he couldn't or didn't want to do those things and asking the participant to please try to find a way to encrypt a copy that he could decrypt.

Each test session lasted for 90 minutes, from the point at which the participant was given the initial task description to the point when the test monitor stopped the session. Manuals for both PGP and Eudora were provided, along with a formatted floppy disk, and participants were told to use them as much as they liked. A more detailed description of the testing process is given in Appendix B.

6.2.2. Participants

The user test was run with twelve different participants, all of whom were experienced users of email, and none of whom could describe the difference between public and private key cryptography prior to the test sessions. The participants all had attended at least some college, and some had graduate degrees. They

[4] This aspect of the test is troublesome in that different test participants were able to extract different amounts of information by asking questions in email, thus leading to test results that are not as standardized as we might like. However, this is in some sense realistic; PGP is being tested here as a utility for secure communication, and people who use it for that purpose will be likely to ask each other for help with the software as part of that communication. We point out also that the purpose of our test is to locate extreme usability problems, not to compare the performance of one set of participants against another, and that while inaccurately improved performance by a few participants might cause us to fail to identify some usability problems, it certainly would not lead us to identify a problem where none exists. Appendix C gives a full report of the email communication for each participant.

represented a mix of ages, genders, education levels and areas of professional expertise. For a detailed description of the participant selection process and resulting demographics, please see Appendix A.

6.3. Results

This is a description of the most significant results we observed from the test sessions, again focusing on the usability standard for PGP that we proposed in section 3.3. Appendix C gives individual transcript summaries for each of the test sessions, and should be referred to for additional results and more detailed information.

- **Avoiding dangerous errors**

  Three of the twelve test participants (P4, P9, and P11) accidentally emailed the secret to the team members without encryption. Two of the three (P9 and P11) realized immediately that they had done so, but P4 appeared to believe that the security was supposed to be transparent to him and that the encryption had taken place. In all three cases the error occurred while the participants were trying to figure out the system by exploring.

  One participant (P12) forgot her pass phrase during the course of the test session and had to generate a new key pair. Participants tended to choose pass phrases that could have been standard passwords, eight to ten characters long and without spaces.

- **Figuring out how to encrypt with any key**

  One of the twelve participants (P4) was unable to figure out how to encrypt at all. He kept attempting to find a way to "turn on" encryption, and at one point believed that he had done so by modifying the settings in the Preferences dialog in PGPKeys. Another of the twelve (P2) took more than 30 minutes[5] to figure out how to encrypt, and the method he finally found required a reconfiguration of PGP (to make it display the PGPMenu inside Eudora). Another (P3) spent 25 minutes sending repeated test messages to the team members to see if she had succeeded in encrypting them (without success), and finally succeeded only after being prompted to use the PGP Plug-In buttons.

- **Figuring out the correct key to encrypt with**

  Among the eleven participants who figured out how to encrypt, failure to understand the public key model was widespread. Seven participants (P1, P2, P7, P8, P9, P10 and P11) used only their own public keys to encrypt email to the team members. Of those seven, only P8, P9 and P10 eventually succeeded in sending correctly encrypted email to the team members before the end of the 90 minute test session, and they did so only after they had received fairly explicit email prompting from the test monitor posing as the team members. P1, P7 and P11 appeared to develop an understanding that they needed the team members' public keys (for P1 and P11, this was also after they had received prompting email), but still did not succeed at correctly encrypting email. P2 never appeared to understand what was wrong, even after twice receiving feedback that the team members could not decrypt his email.

  Another of the eleven (P5) so completely misunderstood the model that he generated key pairs for each team member rather than for himself, and then attempted to send the secret in an email encrypted with the five public keys he had generated. Even after receiving feedback that the team members were unable to decrypt his email, he did not manage to recover from this error.

---

[5] This is measured as time the participant spent working on the specific task of encrypting a message, and does not include time spent working on getting keys, generating keys, or otherwise exploring PGP and Eudora.

- **Decrypting an email message**

  Five participants (P6, P8, P9, P10 and P12) received encrypted email from a team member (after successfully sending encrypted email and publicizing their public keys). P10 tried for 25 minutes but was unable to figure out how to decrypt the email. P9 mistook the encrypted message block for a key, and emailed the team member who sent it to ask if that was the case; after the test monitor sent a reply from the team member saying that no key had been sent and that the block was just the message, she was then able to decrypt it successfully. P6 had some initial difficulty viewing the results after decryption, but recovered successfully within 10 minutes. P8 and P12 were able to decrypt without any problems.

- **Publishing the public key**

  Ten of the twelve participants were able to successfully make their public keys available to the team members; the other two (P4 and P5) had so much difficulty with earlier tasks that they never addressed key distribution. Of those ten, five (P1, P2, P3, P6 and P7) sent their keys to the key server, three (P8, P9 and P10) emailed their keys to the team members, and P11 and P12 did both. P3, P9 and P10 publicized their keys only after being prompted to do so by email from the test monitor posing as the campaign manager.

  The primary difficulty that participants appeared to experience when attempting to publish their keys involved the iconic representation of their key pairs in PGPKeys. P1, P11 and P12 all expressed confusion about which icons represented their public keys and which their private keys, and were disturbed by the fact that they could only select the key pair icon as an indivisible unit; they feared that if they then sent their selection to the key server, they would be accidentally publishing their private keys. Also, P7 tried and failed to email her public key to the team members; she was confused by the directive to "paste her key into the desired area" of the message, thinking that it referred to some area specifically demarcated for that purpose that she was unable to find.

- **Getting other people's public keys**

  Eight of the twelve participants (P1, P3, P6, P8, P9, P10, P11 and P12) successfully got the team members' public keys; all of the eight used the key server to do so. Five of the eight (P3, P8, P9, P10 and P11) received some degree of email prompting before they did so. Of the other four, P2 and P4 never seemed aware that they needed to get the team members' keys, P5 was so confused about the model that he generated keys for the team members instead, and P7 spent 15 minutes trying to figure out how to get the keys but did not succeed.

  P7 gave up on using the key server after one failed attempt in which she tried to retrieve the campaign manager's public key but got nothing back (perhaps due to mis-typing the name). P1 spent 25 minutes trying and failing to import a key from an email message; he copied the key to the clipboard but then kept trying to decrypt it rather than import it. P12 also had difficulty trying to import a key from an email message: the key was one she already had in her key ring, and when her copy and paste of the key failed to have any effect on the PGPKeys display, she assumed that her attempt had failed and kept trying. Eventually she became so confused that she began trying to decrypt the key instead.

- **Handling the mixed key types problem**

  Four participants (P6, P8, P10 and P12) eventually managed to send correctly encrypted email to the team members (P3 sent a correctly encrypted email to the campaign manager, but not to the whole team). P6 sent an individually encrypted message to each team member to begin with, so the mixed key types problem did not arise for him. The other three received an reply email from the test monitor posing as the team member with an RSA key ("Ben"), complaining that he was unable to decrypt their email.

P8 successfully employed the solution of sending that team member an email encrypted only with his own key. P10 explained the cause of the problem correctly in an email to that team member, but didn't manage to offer a solution. P12 half understood, initially believing that the problem was due to the fact that her own key pair was Diffie-Hellman/DSS, and attempting to generate herself an RSA key pair as a solution. When she found herself unable to do that, she then decided that maybe the problem was just that she had a corrupt copy of that team member's public key, and began trying in various ways to get a good copy of it. She was still trying to do so at the end of the test session.

- **Signing an email message**

  All the participants who were able to send an encrypted email message were also able to sign the message (although in the case of P5, he signed using key pairs that he had generated for other people). It was unclear whether they assigned much significance to doing so, beyond the fact that it had been requested as part of the task description.

- **Verifying a signature on an email message**

  Again, all the participants who were able to decrypt an email message were by default also verifying the signature on the message, since the only decryption operation available to them includes verification. Whether they were aware that they were doing so, or paid any attention to the verification result message, is not something we were able to determine from this test.

- **Creating a backup revocation certificate**

  We would have liked to know whether the participants were aware of the good reasons to make a backup revocation certificate and were able to figure out how to do so successfully. Regrettably, this was very difficult to test for. We settled for direct prompting to make a backup revocation certificate, for participants who managed to successfully send encrypted email and decrypt a reply (P6, P8 and P12).

  In response to this prompting, P6 generated a test key pair and then revoked it, without sending either the key pair or its revocation to the key server. He appeared to think he had successfully completed the task. P8 backed up her key rings, revoked her key, then sent email to the campaign manager saying she didn't know what to do next. P12 ignored the prompt, focusing on another task.

- **Deciding whether to trust keys from the key server**

  Of the eight participants who got the team members' public keys, only three (P1, P6, and P11) expressed some concern over whether they should trust the keys. P1's worry was expressed in the last five minutes of his test session, so he never got beyond that point. P6 noted aloud that the team members' keys were all signed by the campaign manager's key, and took that as evidence that they could be trusted. P11 expressed great distress over not knowing whether or not she should trust the keys, and got no further in the remaining ten minutes of her test session. None of the three made use of the validity and trust labeling provided by PGPKeys.

7. Conclusions

7.1. Case study conclusions

Our first conclusion, based largely on the user test results, is that PGP 5.0 is not sufficiently usable to provide effective security for most email users. Our twelve test participants were generally educated and experienced at using email, yet only one third of them were able to use PGP to correctly sign and encrypt an email message when given 90 minutes in which to do so. Furthermore, one quarter of them accidentally exposed the secret they were meant to protect in the process, by sending it in email they thought they had encrypted but had not.

Our second conclusion, based on both the test results and our direct analysis, is that PGP 5.0's failure to be effectively usable is largely due to a mismatch between the design philosophy behind its user interface, and the usability needs of a security utility. PGP's user interface design seems to have prioritized making the basic encryption and signature operations easy to access, and making it convenient to perform the frequent operation of picking keys from the user's key ring for encryption. Making a priority of convenience and obvious access for the most basic and frequently performed operations is a good strategy for a typical application, but PGP would have benefited more from putting the priority on conveying the public key model simply and clearly to the user, making it as obvious as possible when something has been encrypted and when it has not, and making sure that crucial tasks like making a backup revocation certificate were reasonably easy to perform. After all, people who acquire PGP in order to have private and authenticated email can be expected to put some effort into finding the encrypt button if necessary, but they are unlikely to put work into tasks for which they don't understand the need.

7.2. General Conclusions

Security is clearly a key concern of networked computing, and a topic greatly in the vogue. There is tremendous amount of creative, important work in the security techniques in developing new, powerful mechanisms for security. Unfortunately, the same amount of attention has not been lavished on the question of making sure that users actually can and do use the security techniques available to them. To work towards a future where security techniques are effectively and comprehensively used, it is necessary to build systems that are usable by people with a broad variety of backgrounds, training and talents.

We have argued in this paper that security presents a special set of requirements for usability; that it is unforgiving and offers little feedback; and that it is unlikely to be a key motivator for a typical user. Thus, our expectations for usable security software are similarly high, and our analysis is meant to act as a starting point for suggesting further analyses of security software. This is important, not only because of the need to improve the usability of security software, but also because we need to know the usability limits of security software in order to set proper expectations on how it will be used.

More than this, our analysis method suggests a set of open questions in how we design software for usable security applications. We argue that such software must have multiple goals – it should be easy to use and hard to misuse, it should teach users about security, and the interface should grow in sophistication as the user can demonstrate his increased sophistication in applying security. In ongoing work, we are investigating ways to construct security software that satisfies these goals. We hope that our work will lead to security-specific techniques for designing user interfaces that will convey clear and accurate models to users while protecting them from dangerous errors.

8. Related work

We have found very little published research to date on the problem of usability for security. Of what does exist, the most prominent example is the Adage project [9, 16], which is described as a system designed to handle authorization policies for distributed applications and groups. Usability was a major design goal in Adage, but it is intended for use by professional system administrators who already possess a high level of expertise, and as such it does not address the problems posed in making security effectively usable by a more general population. Work has also been done on the related issue of usability for safety critical systems [7], like those which control aircraft or manufacturing plants, but we may hope that unlike the users of personal computer security, users of those systems will be carefully selected and trained.

Aside from Adage, we know only of one paper on usability testing of a database authentication routine [5], and some brief discussion of the security and privacy issues inherent in computer supported collaborative work [13]. John Howard's thesis [3] provides interesting analyses of the security incidents reported to CERT[6] between 1989 and 1995, but focuses more on the types of attacks than on the causes of the vulnerabilities that those attacks exploited, and represents only incidents experienced by entities sophisticated enough to report them to CERT.

References

1. Matt Bishop. *UNIX Security: Threats and Solutions*. Presentation given at SHARE 86.0, Anaheim, CA. March 1996. Available at http://seclab.cs.ucdavis.edu/~bishop/scriv/1996-share86.pdf.

2. Simson Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly and Associates, 1995.

3. John D. Howard. *An Analysis of Security Incidents on the Internet 1989-1995*. Carnegie Mellon University Ph.D. thesis, 1997.

4. John, B. E., & Mashyna, M. M. (1997) Evaluating a Multimedia Authoring Tool with Cognitive Walkthrough and Think-Aloud User Studies. In *Journal of the American Society of Information Science*, 48 (9).

5. Clare-Marie Karat. Iterative Usability Testing of a Security Application. In *Proceedings of the Human Factors Society 33rd Annual Meeting*, 1989.

6. Stephen Kent. Security. In *More Than Screen Deep: Toward Every-Citizen Interfaces to the Nation's Information Infrastructure.* National Academy Press, Washington, D.C., 1997.

7. Nancy G. Leveson. *Safeware: System Safety and Computers*. Addison-Wesley Publishing Company, 1995.

8. Jakob Nielsen. Heuristic Evaluation. In *Usability Inspection Methods*, John Wiley & Sons, Inc., 1994.

9. The Open Group Research Institute. *Adage System Overview*. Published on the web at http://www.osf.org/www/adage/relatedwork.htm, July 1998.

10. Pretty Good Privacy, Inc. *PGP 5.0 Features and Benefits*. Published at http://pgp.com/products/PGP50-fab.cgi, 1997.

11. Pretty Good Privacy, Inc. *User's Guide for PGP for Personal Privacy, Version 5.0 for the Mac OS*. Packaged with software, 1997.

[6] CERT is the Computer Emergency Response Team formed by the Defense Advanced Research Projects Agency, and located at Carnegie Mellon University.

12. Jeffrey Rubin. *Handbook of usability testing: how to plan, design, and conduct effective tests*. Wiley, 1994.

13. HongHai Shen and Prasun Dewan. Access Control for Collaborative Environments. In *Proceedings of CSCW '92*.

14. Cathleen Wharton, John Rieman, Clayton Lewis and Peter Polson. The Cognitive Walkthrough Method: A Practioner's Guide. In *Usability Inspection Methods*, John Wiley & Sons, Inc., 1994.

15. Wogalter, M. S., & Young, S. L. (1994). Enhancing warning compliance through alternative product label designs. *Applied Ergonomics*, 25, 53-57.

16. Mary Ellen Zurko and Richard T. Simon. *User-Centered Security*. New Security Paradigms Workshop, 1996.

A. Description of test participants

A.1. Recruitment

The test participants were recruited through advertising posters on the CMU campus and posts on several local newsgroups (cmu.misc.market, pgh.general, and pgh.jobs.offered) with the exception of P11 and P12, who were recruited through personal contacts. The text of the poster and newsgroup posts read:

> Earn $20 and help make computer security better!
>
> I need people to help me test a computer security program to see how easy it is to use. The test takes about 2 hours, and should be fun to do.
>
> If you are interested and you know how to use email (no knowledge of computer security required), then call Alma Whitten at 268-3060 or email alma@cs.cmu.edu.

More than 75 people responded to the advertisements, and 38 people completed the background interview. From those 38, we disqualified 8 who already had some knowledge of public key cryptography. We then chose 12 participants based on age, gender, education level and area of education/work experience, trying for the widest and most evenly distributed range available.

A.2. Participant demographics

This table describes the demographic distribution of the twelve test participants:

Gender	6 female 6 male
Age	3 age 25 or younger 3 age 26 to 35 3 age 36 to 45 3 age 45 or older
Highest education level[7]	2 had some college 4 had undergraduate degrees 4 had some graduate school 2 had graduate degrees
Education or career area[8]	2 did computer programming 4 did biological science (pharmacy, biology, medicine) 4 did humanities or social science (education, business, sociology, English) 2 did fine arts (graphic design)

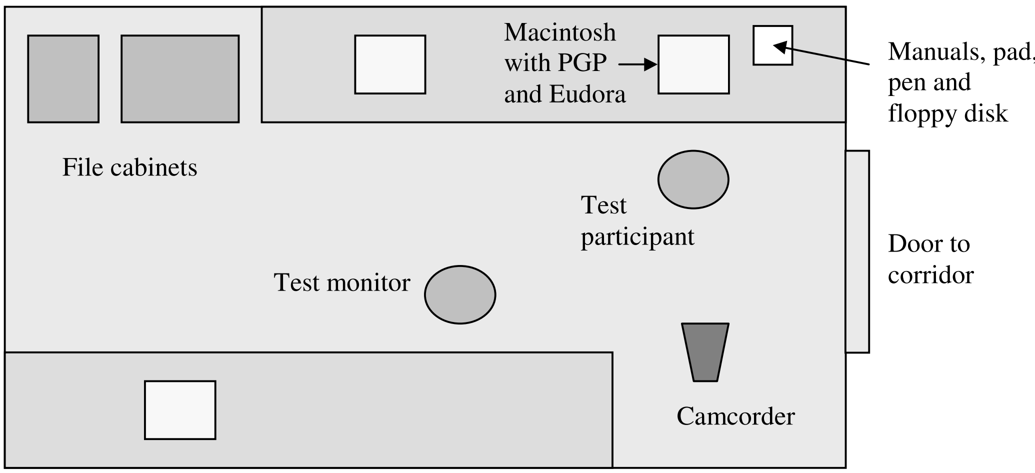
[7] The original test plan called for some participants with only a high school education, but none responded to the advertisements.

[8] This categorization was biased toward familiarity with computer programming and then toward training in hard science, so that the psychology grad student who works as a research programmer is classified as a computer programmer, and the business/biology major is classified as having a biological science background.

B. Description of testing process

B.1. Test environment

The testing was done in a small lab set up as depicted below.



PGP and Eudora were both installed on the Macintosh, and Eudora was configured to properly access an email account set up for test participants, and to use the PGP Plug-In. PGP was put into the state it would be in right after installation, with one exception: we deleted all the keys that ordinarily come with PGP and let the participants begin with an empty key ring. We put complete printouts of the PGP and Eudora manuals into labeled 3-ring binders and pointed them out to the participants at the beginning of the test sessions.

Our test set-up had one additional peculiarity worth mentioning, which is that in our efforts to remove extraneous programs that might distract the participants during the test, we inadvertently removed SimpleText, which meant that test participants were unable to read the “Phil's Letter,” “PGPFreeware 5 README,” and “QuickStart” documents. As we described in our analysis, we strongly doubted that those three documents had the potential to be of much help to novice users, so we chose to maintain consistency over the test sessions and did not restore SimpleText for later participants.

B.2. Greeting and orientation

Each participant was met in the building lobby, and escorted from there to the lab for the test session.

The orientation for the test had four components:
1. The participant was given two copies of the consent form[9], and asked to read it and sign one copy.
2. The written briefing was read aloud to the participant[10], and then the written document was given to the participant for his or her own reference during the test. This briefing explained the following:
    a. that they were helping test Eudora and PGP, not being tested themselves;
    b. that it would be extremely helpful if they could “think aloud” as much as possible during the test;
    c. that the premise of the test was that they were volunteering for a political campaign, and that their task would be to send email updates to the members of the campaign team, using encryption and digital signatures to ensure that the email was kept secret and wasn’t forged;

[9] See Appendix D.
[10] See Appendix E.

    d. what their email address and password would be for the purposes of the test;
    e. that Eudora and PGP were already installed, and that the manuals, pad, pen and floppy disk were there for them to use as much as they liked;
    f. that they'd be given a 5 minute tutorial on basic use of Eudora before we began the actual testing.
3. They were given the 5 minute Eudora tutorial, and it was verified that they understood how to use it for sending and receiving email.
4. The initial task description was read aloud to the participant[11], and then they were given the written document for their own use. This document gave them the following information:
    a. Names and email addresses for the campaign manager and four members of the campaign team;
    b. The text of a message giving a series of speaking dates and locations for the candidate;
    c. A request that they please use PGP and Eudora to send the message in a secure, signed email to the campaign manager and all the other campaign team members;
    d. A further request to then wait for any email responses from the team members and to follow any instructions they might give.

B.3. Testing

The participant's actions during the actual testing were recorded both by the camcorder, which was focused on the monitor screen, and by the test monitor. During the test the test monitor was seated as shown in the test environment diagram, about six feet away from the participant, behind them and to the side. The test monitor used a laptop computer equipped with a WaveLAN network connection both to take monitoring notes and to remotely play the roles of the various campaign team members (by reading and replying to the participant's email) as necessary.

The original test design called for the testing to be done in two parts: a 45 minute session with the task and test scenario described above, to be followed by a debriefing and then another 45 minute session in which the test monitor would directly ask the participant to try to perform specific tasks, such as revoking their public key or signing someone else's public key. In practice, we quickly found that most of the participants could not succeed at the initial task of sending signed and encrypted email within 45 minutes, and that it made more sense to let them continue with that task for up to the full 90 minutes. When a participant did succeed at the initial task, we found it seemed more natural to have the fictional campaign manager prompt them by email to attempt additional tasks, rather than to stop the test and start a second session in which the test monitor prompted them directly. In only one case did we follow the original test design, with a participant (P6) who did succeed at the initial task within 45 minutes.

In order to succeed at the initial task of sending signed and encrypted email to all the members of the campaign team, the participant needed to accomplish the following:

1. Generate a key pair of their own.
2. Make their public key available to the campaign team members, either by sending it to the key server, or by emailing it to them directly.
3. Get the campaign team members' public keys, either by fetching them from the key server or by sending email directly to the team members to request their public keys.
4. Encrypt the secret message using the team members' public keys, sign it using their own private key, and send it.

Email that was encrypted with the wrong key(s) caused the participant to get replies from the team members complaining that they couldn't decrypt the message; repeated occurrences caused the team members to also ask if the participant was using their keys to encrypt.

[11] See Appendix C.

If the participant succeeded at the initial task, they received several email responses:

1. A signed and encrypted email from the campaign manager giving them an update to the secret message; this tested whether they could decrypt and read the email successfully.
2. An email from the campaign manager reminding them to back up their key rings and make a backup revocation certificate; this tested whether they could do those things.
3. An email from a member of the campaign team whose key pair was RSA rather than Diffie-Hellman/DSS, complaining that he was unable to decrypt their email; this tested whether they could identify the mixed key types problem and use the correct solution of sending that team member email encrypted only with his own public key.

B.4. Debriefing

After stopping the test, the test monitor turned off the camcorder and turned on the audio tape recorder, and then verbally asked the participant the questions in the document titled *Questionnaire to follow part one of PGP Usability Test*[12]. The test monitor then thanked the participant for their help, paid them the $20 in cash, and ended the test session.

C. Summaries of test session transcripts

Times are rounded to nearest 5 minutes. "Maria" is the fictional campaign manager, "Ben" and "Paul" are fictional campaign team members (and Ben's key is RSA rather than Diffie-Hellman/DSS); email from them is actually sent by the test monitor using a laptop as a remote link.

P1: male, age 29, grad degree in education, now university administrator

00:00 to 00:05	Typed in the secret email message.
00:05 to 00:10	Tried to figure out how to encrypt his message, explored PGP.
00:10 to 00:15	Generated a key pair for himself.
00:15 to 00:20	Read manual, focused on how to get other people's public keys.
00:20 to 00:25	Backed up his key rings on the floppy disk.
00:25 to 00:35	Sent his public key to the key server.
00:35 to 00:40	Sent email to the team members asking for their public keys, but encrypted it with his own public key.
00:40 to 00:45	Got email from the test monitor posing as Maria, saying she can't decrypt his email.
00:45 to 00:50	Sent email to team members, encrypted with his own public key again.
00:50 to 00:55	Got email from the test monitor posing as Maria, saying she still can't decrypt his email, and enclosing her public key.
00:55 to 01:20	Tried to import Maria's public key from her email message.
01:20 to 01:25	Fetched team members' public keys from the key server.
01:25 to 01:30	Tried to figure out whether to trust the public keys from the key server.

Comments:

- He was able to successfully generate his own key pair, send his public key to the key server, and get the team members' public keys from the key server.
- He tried (for 25 minutes) and failed to import a key from an email message.
- He did not successfully send signed and encrypted email to the team members before the end of the 90 minute test session.
- He did also successfully back up his key rings onto a floppy disk.

[12] See Appendix D.

- When preparing to send his key to the key server, he expressed worry that he couldn't tell whether the key in the PGPKeys display was his public key or his private key.
- When trying to request the team members' public keys by email, he didn't understand that encrypting with his own key (only) would prevent them from being able to read his message.
- When trying to get Maria's public key out of her email message and into his key ring, he copied the key onto the clipboard and then repeatedly tried to decrypt it, rather than using the Import Key command or simply pasting it into PGPKeys.

P2: male, age 38, IS major with some grad school, now database programmer

00:00 to 00:05	Set up Eudora mail aliases for the campaign team members.
00:05 to 00:20	Looked at manuals and explored PGP.
00:20 to 00:25	Generated a key pair for himself.
00:25 to 00:55	Tried to figure out how to encrypt an email message. Reconfigured PGP to display the PGPMenu on the Eudora menu bar.
00:55 to 01:00	Sent his public key to the key server.
01:00 to 01:05	Sent email to team members, but encrypted it with just his own public key.
01:05 to 01:10	Got email from the test monitor posing as Maria, saying that she can't decrypt his email.
01:10 to 01:20	Again sent email to team members encrypted with just his own public key, but this time also signed the message after encrypting it.
01:20 to 01:30	Got another email from the test monitor posing as Maria, saying that she still can't decrypt his email; didn't make any further progress.

Comments:

- He was able to successfully generate his own key pair and send his public key to the key server.
- He was unable to discover any of the direct routes to encrypting his email (using the PGP Eudora plug-in buttons or pull-down menu, or using the clipboard with PGPTools) and instead used the Preferences dialog in PGP to add the generalized PGPMenu to the Eudora menu bar.
- He did not successfully get any of the team members' public keys; in fact, he never appeared to realize that he needed to do so.
- He did not successfully send signed and encrypted email to the team members before the end of the 90 minute test session.
- He never appeared to understand the need to exchange public keys, or to have a clear sense of how the keys were to be used.

P3: female, age 49, grad degree, business/biology major, now computer operator

00:00 to 00:05	Generated a key pair for herself.
00:05 to 00:15	Sent a plain text email to the team members asking for their public keys.
00:15 to 00:20	Got email from the test monitor posing as Maria, with Maria's public key and the suggestion to get the other team members' public keys from the key server.
00:20 to 00:55	Tried to get the other team members public keys from the key server and eventually succeeded.
00:55 to 01:05	Sent a test email message to the team members to see if she'd successfully encrypted and signed it (no).
01:05 to 01:10	Got email from the test monitor posing as Maria, telling her the test email wasn't signed or encrypted.
01:10 to 01:15	Sent another test message, but still wasn't successful at signing or encrypting it.

01:15 to 01:20	Got email from the test monitor posing as Maria, telling her that the test message still wasn't signed or encrypted, and asking if she's pushing the encrypt and sign buttons on the message before sending it.
01:20 to 01:25	Sent signed and encrypted email to Maria successfully.
01:25 to 01:30	Sent her public key to the key server (after a prompt from the test monitor posing as Maria).

Comments:

- She was able to successfully generate her own key pair and send her public key to the key server.
- With some prompting from the test monitor posing as Maria, she was able to get the team members' public keys from the key server, and finally send correctly signed and encrypted email to the team members.
- Although she asked for the team members' public keys via email, she did not, or was not able to, import Maria's key from the email message; instead she spent 35 minutes figuring out how to fetch the public keys from the key server.
- She didn't manage to find the Sign and Encrypt plug-in buttons on her own, nor did she figure out (in 25 minutes of working on it) any of the alternative ways to get her message signed and encrypted.

P4: male, age 39, undergrad degree in English, now writer

00:00 to 00:05	Sent the secret message to the team members in a plain text email.
00:05 to 00:10	Got email from the test monitor posing as Maria, pointing out the error, reiterating the importance of signing and encrypting, and asking him to try to send a signed and encrypted test message before going any further.
00:10 to 00:20	Tried to figure out how to encrypt; looked at manual and opened PGP.
00:20 to 00:25	Generated a key pair for himself.
00:25 to 00:30	Quit PGPKeys and saved backup of his key rings (in the PGP folder).
00:30 to 00:40	Continued trying to figure out how to encrypt, sent a test email message.
00:40 to 00:45	Got email from the test monitor posing as Maria, telling him the test message wasn't signed or encrypted either.
00:45 to 01:05	Continued trying to figure out how to encrypt, sent another test message but seemed to already know he hadn't succeeded in signing or encrypting it.
01:05 to 01:20	Continued trying to figure out how to encrypt. Sent another test message after modifying the settings in the PGPKeys Preferences dialog.
01:20 to 01:25	Got email from the test monitor posing as Maria, telling him his test message still isn't signed or encrypted, and asking him to please keep trying even though it must be frustrating.
01:25 to 01:30	Continued trying to figure out how to encrypt, without success.

Comments:

- He was able to successfully generate his own key pair.
- He accidentally sent the secret message in a plain text email.
- He was not able to figure out how to encrypt and sign his email message within the 90 minute test session (the test monitor posing as Maria didn't prompt him beyond offering encouragement).
- He never appeared aware of the need to get the team members' public keys or the need to make his own public key available.
- He did back up his key rings, but did so within the same folder as the originals.
- He seemed to expect to be able to "turn on" encryption and then have it happen invisibly; at one point toward the end he thought he had done so by modifying the settings in the PGPKeys Preferences dialog. Furthermore, it appeared that his expectation that it would be invisible caused him to have no sense of whether or not he had managed to sign or encrypt his messages.

- He seemed to know he needed keys for the team members (after encountering the "please drag recipients from this list" dialog) but appeared to think that the keys he needed must be around somewhere in PGP and that he was just having trouble finding them.

P5: male, age 47, sociology major with some grad school, now clerical worker

00:00 to 00:05	Typed the secret message into Eudora and saved it in a file.
00:05 to 00:15	Tried to sign and encrypt his saved file using PGPTools, cancelled when he got to the dialog that asks for the public keys to encrypt with.
00:15 to 00:20	Repeated the above.
00:20 to 00:25	Read the manual, realized he needed to generate keys.
00:25 to 00:35	Generated a key pair for each of the campaign team members, not for himself.
00:35 to 00:45	Signed and encrypted his saved file using PGPTools, seemed to think that completing this process caused his email message to be sent.
00:45 to 00:50	Exported his five key pairs to a file.
00:50 to 00:55	Checked for new mail, wondered aloud if his email was sent. I intervened to show him how to tell if a Eudora message was sent or not.
00:55 to 01:05	Used the Eudora plug-in menu to access the PGP plug-in and signed and encrypted his email message using the five key pairs, then sent it to the team members.
01:05 to 01:10	Got email from the test monitor posing as Maria, saying she can't decrypt his message and asking if he used her public key to encrypt it. Figured out that something's wrong.
01:10 to 01:30	Experimented with trying to send himself encrypted email, but couldn't figure out what the problem was. Eventually seemed to conclude that the PGP plug-in must not be installed.

Comments:

- He was unable even to generate a key pair for himself, since he mistakenly thought he needed to generate key pairs for each of the campaign team members instead.
- He never figured out that he needed to get pre-existing public keys for the campaign team members.
- He never figured out that he needed a key pair of his own, much less that he would need to make his public key available to the campaign team members.
- He verbally expressed a lot of worry about the security of the keys stored on the hard drive.
- He did not succeed at sending the signed and encrypted message to the members of the campaign team within the 90 minute test session, nor did he succeed at any of the prerequisite tasks.
- He clearly did not understand the basic model of public key cryptography and key distribution, and his understanding did not seem to increase over the course of the test session. He understood that he needed keys for the people he wanted to send encrypted email to, but apparently nothing beyond that.
- He had some trouble figuring out how to get his message signed and encrypted, which appeared to eventually be compounded by his attributing Maria's inability to decrypt his email to a problem in the signing and encryption process rather than a problem with the keys used.

P6: male, age 31, psychology grad student, also research programmer

00:00 to 00:05	Typed message into Eudora, tried to sign and encrypt it using the PGP plug-in buttons, cancelled when he got to the dialog that asks for public keys to encrypt with.
00:05 to 00:15	Figured out that he needed to generate a key pair for himself and did so.
00:15 to 00:20	Backed up his key rings on the floppy disk.
00:20 to 00:25	Tried again to sign and encrypt his message, dragged his own public key to the recipients' list, then realized he needed the team members' public keys and cancelled.
00:25 to 00:30	Sent his public key to the key server.
00:30 to 00:35	Fetched the team members' public keys from the key server.
00:35 to 00:40	Noted all the team members' public keys are signed with Maria's private key, decided it was okay to trust them.
00:40 to 00:45	Sent the secret message to each team member in an individually encrypted and signed email.

At this point the test monitor stopped the test and debriefed, then proceeded to ask him to perform specific tasks directly, following the original test design.

00:45 to 00:50	The test monitor asked him to send a signed and encrypted email to Paul and Ben, to see what he'd do about the mixed key types warning. He sent the message despite the warning, commenting that they could always send him email if there was a problem.
00:50 to 01:00	The test monitor, posing as Maria, sent him a signed and encrypted message to see if he could decrypt and verify it; he had some initial trouble getting the results after decryption, but succeeded.
01:00 to 01:15	The test monitor asked him to create a backup revocation certificate; he made a test key pair and then revoked it. He thought that fulfilled the task, so the test monitor went on.
01:15 to 01:20	The test monitor asked him to label Maria's public key as completely trusted. PGP wouldn't let him do that, since her public key had not been signed by some completely trusted public key.
01:20 to 01:25	The test monitor asked him to sign Maria's public key. PGP wouldn't let him sign it because he had no default key pair set (as a result of having generated and revoked that test key pair), but didn't tell him that was the problem, so he gave up.
01:25 to 01:30	The test monitor asked him to revoke his public key. He did so, but didn't send the revocation to the key server.

Comments:

- He successfully generated a key pair for himself, sent his public key to the key server, got the campaign team members' public keys from the key server, and correctly signed, encrypted and sent the secret message, all in the first 45 minutes of the test session.
- He sent an individual signed and encrypted message to each member of the campaign team, so he didn't encounter the mixed key types warning until the test monitor made him do so in the second half of the test.
- He backed up his key rings onto the floppy disk.
- He understood that the public keys he retrieved were signed with Maria's key and stated that as evidence they could be trusted.
- He successfully decrypted and verified a message from the test monitor posing as Maria, although it was unclear how aware of the signature verification he was.
- Evaluating his understanding of revocation is problematic, since we weren't really able to fit it believably into the test scenario; he might not have publicized the revocation at the end simply because he didn't think that's what the the test monitor was asking for.
- It looked like he would have been able to sign Maria's key easily except for the unexpected default key problem (see below).

- Creating and revoking a test key pair caused him to have no default key pair set; PGP then refused to let him sign a key, and offered no explanation for the refusal nor any information to alert him that he needed to set a new default key pair.

P7: female, age 40, undergrad degree in biology, now clerical worker

00:00 to 00:20	Explored, tried to figure out how to sign and encrypt.
00:20 to 00:25	Generated a key pair for herself.
00:25 to 00:30	Tried to figure out how to distribute her public key.
00:30 to 00:40	Tried to figure out how to paste her public key into an email message.
00:40 to 00:50	Sent email to team members encrypted just with her own public key.
00:50 to 00:55	Sent her public key to the key server.
00:55 to 01:00	Continued trying to figure out how to email her public key to the team members.
01:00 to 01:05	Sent her public key to the team members in a plain text email.
01:05 to 01:20	Tried to figure out how to get the team members' public keys.
01:20 to 01:25	Tried to figure out how to back up her key rings.
01:25 to 01:30	Tried to figure out how to sign and encrypt.

Comments:

- She successfully generated a key pair for herself, sent her public key to the key server, and emailed her public key to the members of the campaign team.
- She was not able to find a way to get the team members' public keys, and this prevented her from being able to send encrypted email.
- She seemed to understand the basic public key model, but was unsure of the validity of her understanding, and was easily put off and confused by small errors.
- She appeared to confuse the "please drag recipients" encryption dialog box with an address book, not realizing that it was prompting her for keys for the recipients.
- She was confused by the manual directive to paste her key into the "desired area" of the email message, thinking that it specified some exact location that she was unable to find.
- When her initial attempt to fetch Maria's key from the key server failed (due to mis-typing?) she took that as evidence that she was on the wrong track and never tried again.

P8: female, age 20, undergrad student, business major

00:00 to 00:05	Explored, generated a key pair for herself.
00:05 to 00:10	Looked at PGPKeys display, read manual.
00:10 to 00:15	Sent email message to team members encrypted just with her own public key.
00:15 to 00:20	Got email from the test monitor posing as Maria, saying she can't decrypt that email.
00:20 to 00:25	Tried sending the email again, still encrypted just with her own public key.
00:25 to 00:30	Got email from the test monitor posing as Maria, saying she still can't decrypt it, and asking if she's using Maria's public key.
00:30 to 00:35	Sent her public key in an email, still encrypted just with her own public key.
00:35 to 00:40	Got email from the test monitor posing as Maria, saying she still can't decrypt it, and asking if she needs to get Maria's public key.
00:40 to 00:45	Fetched team members' public keys from the key server after referring to manual.
00:45 to 00:50	Sent secret to team members in signed and encrypted email.
00:50 to 00:55	Got email from the test monitor posing as Maria, requesting an update to the secret.
00:55 to 01:00	Got email from the test monitor posing as Ben, saying he can't decrypt her email, sent him a message saying "Your key is blue! Let me see what I should do."

01:00 to 01:05	Decrypted Maria's message and sent the updated secret to the team members in signed and encrypted email. Didn't appear to react to the mixed key types warning.
01:05 to 01:10	Got email from the test monitor posing as Ben, saying he can't decrypt that one either.
01:10 to 01:15	Sent email to Ben telling him the problem is that his key is RSA, and that he should update his copy of PGP.
01:15 to 01:20	Got email from the test monitor posing as Ben, saying that he can't update right now and asking her to find a way to send him email that he can decrypt. Sent him an email encrypted just with his public key.
01:20 to 01:30	Got email from the test monitor posing as Maria, reminding her to back up her key rings and make a backup revocation certificate. Backed up her key rings and then revoked her key; sent email saying that she made a backup but couldn't figure out how to do the backup revocation certificate.

Comments:

- She was able to generate a key pair for herself and to send her public key to the team members via email.
- She figured out that she needed to get keys for the team members only after three successively stronger hints from the test monitor posing as Maria, but then was able to figure out how to get the keys quickly and easily.
- She was able to send signed and encrypted email to the team members once she understood that she needed their public keys.
- She was able to figure out why Ben couldn't decrypt her message and find the solution to the problem.
- She was able to decrypt and read Maria's message easily.
- She was able to back up her key rings when prompted to do so.
- She didn't understand that she needed to use the team members' public keys to encrypt until she'd received multiple explicit prompts from the test monitor posing as Maria.
- She didn't understand the directive to make a backup revocation certificate, and it might have taken her a while to recover from the results of her attempt to do so.

P9: female, age 24, medical student

00:00 to 00:05	Emailed the secret to the campaign team members in plain text.
00:05 to 00:10	Got email from the test monitor posing as Maria, pointing out the error, reiterating the importance of signing and encryption, and asking her to send a signed and encrypted test message before going any further.
00:10 to 00:30	Tried to sign and encrypt a test message, got stuck at the dialog that asks for the public keys to encrypt with.
00:30 to 00:35	Generated a key pair for herself. Sent a test message encrypted with just her own public key.
00:35 to 00:40	Got email from the test monitor posing as Maria, saying she can't decrypt that and asking if she's using Maria's key to encrypt.
00:40 to 00:45	Sent two more messages encrypted just with her own public key. Got another email from the test monitor posing as Maria, saying she can't decrypt those and asking if she's using Maria's key to encrypt.
00:45 to 00:50	Tried to figure out how to get Maria's public key.
00:50 to 00:55	Fetched Maria's public key from the key server. Sent a signed and encrypted test message.
00:55 to 01:00	Got email from the test monitor posing as Maria, saying that was good work and reminding her to give Maria her public key. Emailed her public key to Maria.
01:00 to 01:05	Got signed and encrypted email from the test monitor posing as Maria, with an updated secret.

01:05 to 01:10	Sent email to Maria asking if the block of text in the last message is a key.
01:10 to 01:15	Got email from the test monitor posing as Maria, saying she didn't send a key and that the block is just the message.
01:15 to 01:20	Decrypted Maria's message, sent email saying the updated secret is on the way.
01:20 to 01:25	Sent updated secret to all team members encrypted just with Maria's public key. Got email from the test monitor posing as Paul, saying he can't decrypt that message.
01:25 to 01:30	Sent updated secret to all team members encrypted just with Maria's public key and her own public key, then began fetching the other team members' public keys from the key server.

Comments:

- She sent the secret in plain text initially, but realized her error before being told.
- She was able to generate a key pair for herself successfully.
- She was able to get Maria's key and send signed and encrypted email successfully after two fairly explicit prompts from the test monitor posing as Maria.
- She was able to send her key to Maria in email after being prompted by the test monitor, posing as Maria, to give Maria her key.
- She accidentally sent the secret in a plain text email.
- She didn't understand the public key model well: she tried sending email to Maria encrypted only with her own key until the test monitor, posing as Maria, repeatedly prompted her to get Maria's key, and then after successfully sending signed and encrypted email to Maria, she tried to send signed and encrypted email to the whole team using only her key and Maria's.
- She mistook the encrypted block she received in email for a key.

P10: male, age 45, some undergrad work in pharmacy, now does human resources

00:00 to 00:15	Generated a key pair for himself, looked at it in PGPKeys, experimented with generating another key pair but then cancelled.
00:15 to 00:25	Emailed the secret to Paul, encrypted just with his own public key.
00:25 to 00:30	Got email from the test monitor posing as Paul, saying he can't decrypt that, and asking if he used Paul's key to encrypt.
00:30 to 00:35	Fetched team members' public keys from the key server. Backed up his key rings.
00:35 to 00:40	Sent the secret to the team members in a signed and encrypted email.
00:40 to 00:45	Got email from the test monitor posing as Maria, thanking him and reminding him that now they need his public key.
00:45 to 00:50	Got email from the test monitor posing as Ben, saying he can't decrypt that message.
00:50 to 00:55	Sent email to Ben correctly explaining the key type problem.
00:55 to 01:00	Emailed his public key to the team members. Got email from the test monitor posing as Ben, saying his copy of PGP won't do DSS keys, and asking him to send a copy that Ben can decrypt with his RSA key.
01:00 to 01:05	Got signed and encrypted email from the test monitor posing as Maria, thanking him for sending his key and giving him an updated secret to send out.
01:05 to 01:30	Tried to figure out how to decrypt Maria's email.

Comments:

- He was able to generate a key pair for himself successfully.
- He initially sent the secret encrypted only with his own key.
- After prompting, he was able to get the team members' keys from the key server.
- He was able to send his public key to the team members via email.
- He didn't figure out how to send email that Ben could decrypt.
- He was unable to figure out how to decrypt Maria's email within the 25 minutes before the test ended.

- Initial trouble with sending email encrypted only with his own key.
- Bothered when PGP didn't match some of the team members' keys with their grayed out representations in the "please drag recipients" dialog.
- Didn't figure out that he should send Ben email encrypted only with Ben's key.
- Didn't figure out how to decrypt an encrypted message in 25 minutes of trying.

P11: female, age 33, undergrad degree in fine arts, now graphic designer

00:00 to 00:05	Sent the secret out in a plain text email, but realized the error on her own.
00:05 to 00:10	Got email from the test monitor posing as Maria, reiterating the importance of signing and encryption and asking her to send a signed and encrypted test message.
00:10 to 00:20	Generated a key pair for herself.
00:20 to 00:25	Tried to figure out how to distribute her public key and get the team members' public keys.
00:25 to 00:30	Tried to figure out how to back up her key rings. Sent her public key to the key server.
00:30 to 00:40	Emailed her public key to Maria.
00:40 to 01:00	Tried to figure out how to encrypt. Sent email to team members encrypted just with her own public key.
01:00 to 01:05	Got email from the test monitor posing as Maria, saying she can't decrypt that and asking if she used Maria's key to encrypt.
01:05 to 01:10	Sent email to Maria asking for Maria's public key.
01:10 to 01:20	Fetched the team members' public keys from the key server. Read about trusting keys and checking fingerprints.
01:20 to 01:25	Got email from the test monitor posing as Maria, with Maria's public key.
01:25 to 01:30	Worried about how to check the validity of the team members' public keys.

Comments:

- She sent the secret in plain text initially, but realized her error without being told.
- She was able to generate a key pair for herself, send her public key to the key server, send her public key in an email message, and fetch the team members' public keys from the key server.
- She initially encrypted her email to the team members with just her own key.
- She figured out that Ben's key was blue because it was an RSA key.
- She didn't successfully send signed and encrypted email because she was afraid to trust the keys she got from the key server.
- Bothered by not being able to figure out which of the icons in the PGPKeys display was her public key and which was her private key; afraid of accidentally sending her private key.
- Initial trouble with sending email encrypted only with her own key.
- Afraid that sending her key to the key server had failed because all she got was the "receiving data…" message.
- Confused the Eudora signature button with the PGP plug-in signature button.
- Worried that the PGP plug-in buttons weren't connected to anything because nothing seemed to happen when she clicked them.
- Nervous about publicizing her public key, it seemed to be at odds with her expectation that keys need to be kept secret, was afraid that she was misunderstanding and making a mistake.
- Too afraid of making a mistake to trust the keys that she got from the key server, alarmed by the default "untrusted" key properties, didn't appear to notice that the keys were all signed by Maria.

P12: female, age 22, undergrad degree in fine arts, now graphic designer

00:00 to 00:10	Generated a key pair for herself.
00:10 to 00:15	Sent her public key to the key server.
00:15 to 00:20	Created an email message and pasted her public key into it. Fetched team members' public keys from the key server.
00:20 to 00:25	Created another email message and typed the secret into it.
00:25 to 00:45	Sent the secret to the team members in a signed and encrypted email.
00:45 to 00:50	Got signed and encrypted email from the test monitor posing as Maria, reminding her to back up her key rings and make a backup revocation certificate. Decrypted it.
00:50 to 00:55	Got email from the test monitor posing as Ben, saying he can't decrypt her email. Decided it's because his public key is a different type from hers.
00:55 to 01:00	Sent email to Ben asking if he can create a new key pair for himself.
01:00 to 01:05	Tried to generate a RSA key pair for herself so that her key would be the same type as Ben's (PGP wouldn't let her). Tried changing the validity and trust settings on Ben's public key.
01:05 to 01:10	Got email from the test monitor posing as Ben, saying there's nothing wrong with his key pair and he doesn't want to generate a new one right now. Sent Ben email asking if he has her public key and if they can set up a file somewhere so that she can import his public key.
01:10 to 01:20	Got email from the test monitor posing as Ben, giving her his public key and saying that he has hers. Repeatedly copied Ben's public key from the email and pastes it into her key ring (PGPKeys) but assumed it wasn't working because the display didn't change.
01:20 to 01:25	Sent email to Ben saying she's stuck.
01:25 to 01:30	Tried to decrypt Ben's public key block.

Comments:

- She successfully generated a key pair for herself, sent her public key to the key server, pasted her public key into an email message, fetched the team members' public keys from the key server, sent the secret to the team members in a signed and encrypted email, and decrypted and read Maria's reply.
- She figured out that Ben couldn't decrypt because his key was RSA, but wasn't able to figure out the solution to the problem.
- She was bothered by not being able to tell which icon in PGPKeys represented her public key and which her private key, and afraid to send her key to the key server for fear of accidentally sending her private key.
- She decided after experimentation that the key pair icon was her public key, and the icons below it (the signature info) were her private key.
- Concluded erroneously that the PGP plug-in wasn't installed, and used PGPTools and the clipboard instead.
- Forgot her initial pass phrase and had to generate and publicize a second key pair.
- Initially understood why Ben couldn't decrypt her message, but went on to a series of erroneous explanations while trying to figure out a solution.

D. Consent form

CARNEGIE MELLON UNIVERSITY
CONSENT FORM

Project Title: PGP 5.0 Usability Test
Conducted By: Alma Whitten, Computer Science Department

I agree to participate in the observational research conducted by students under the supervision of Dr. Doug Tygar. I understand that the proposed research has been reviewed by the University's Institutional Review Board, and that to the best of their ability they have determined that the observations involve no invasions of my rights or privacy, nor do they incorporate any procedure or requirements which may be found morally or ethically objectionable. I understand that my participation is voluntary, and that if at any time I wish to terminate my participation in this study, I have the right to do so without penalty. I understand that I will be paid $20 for my participation when I have completed the experiment.

If you have any questions about this study, you should feel free to ask them now or at any time during the experiment, or later by contacting:

Professor Doug Tygar
CMU School of Computer Science
412-268-6340
tygar@cs.cmu.edu

You may report any objections to this study, either orally or in writing, to:

Susan Burkett
Associate Provost
Carnegie Mellon University
412-268-8746

Purpose of the study: I understand that I will be using the email program Eudora and the security program PGP. I understand that I will be asked to send and receive secure electronic mail, and that I may be asked to use PGP to perform additional security management tasks. I realize that the email messages I send and receive during the course of this test will be saved for analysis, and that I will be asked to reveal any passwords or pass phrases that I use during the course of the testing. I am aware that this is an evaluation of the design of PGP's user interface, and not an evaluation of my own skills or competence.

I understand that the following procedure will be used to maintain my anonymity in analysis, publication and/or presentation of any results. Names will not be recorded; instead, each participant will be assigned a number. The researchers will save the data, videotapes and audiotapes by participant number, not by name. Only members of the research team will view or listen to the tapes in detail. No other researchers will have access to these tapes.

Optional Permission: I understand that the researchers may want to use a short portion of videotape for illustrative reasons in presentations of this work. I give my permission to do so provided that my name and face will not appear.
_______ YES _______ NO Please initial here _______

I understand that in signing this consent form, I give Professor Tygar and his associates permission to present this work in written and oral form, without further permission from me.

______________________________________ ______________________________________
Name (please print) Signature

______________________________________ ______________________________________
Telephone Date

E. Initial briefing document

What you need to know

This is a test of the design of PGP and of PGP as an addition to the email program Eudora. You are not being tested; you are helping me test PGP. At some points you may feel frustrated and stuck, but please do keep trying and don't feel bad, because seeing where people get stuck and what they do to get unstuck is exactly the kind of data I need from this testing.

If you can manage it, it is extremely useful to me if you "think aloud" during the test. The camcorder has a microphone that will pick up what you say, and I'll be taking notes as well. The more informative you can be about what you are doing and thinking, the better my data will be.

The scenario for the first part of the test is that you are volunteering for a political campaign, and the role that you have been given is that of Campaign Coordinator.
Your task is to send updates about the campaign plan out to the members of the campaign team by email. It is very important that the plan updates be kept secret from everyone other than the members of the campaign team, and also that the team members can be sure that the updates they receive haven't been forged. In order to ensure this, you and the other team members will need to use PGP to encrypt and digitally sign your email messages.

Your email address for the purposes of this test is ccoord@wanton.trust.cs.cmu.edu, and your password is **volnteer**. You should use the title "Campaign Coordinator" rather than using your own name.

Eudora and PGP have both been installed, and Eudora has been set up to access your email account. Manuals for both Eudora and PGP are in the black binders to your right; use them as much as you like. The pad, pens, and floppy disk are also there for you to use if you want them.

Before we start the test itself, I'll be giving you a very basic demonstration of how to use Eudora. The goal is to have you start out the test as a person who already knows how to use Eudora to send and receive email, and who is just now going to start using PGP as well to make sure your email can't be forged or spied on while it's being delivered over the network. The Eudora tutorial will take about 5 minutes, and then we'll begin the actual testing.

F. Initial task description

The campaign manager is Maria Page, mpage@wanton.trust.cs.cmu.edu.

The other members of the campaign team are:

Paul Butler, butler@wanton.trust.cs.cmu.edu
Ben Donnelly, bend@wanton.trust.cs.cmu.edu
Sarah Carson, carson@wanton.trust.cs.cmu.edu
Dana McIntyre, dmi@wanton.trust.cs.cmu.edu

Please use PGP and Eudora to send the following message in a secure, signed email to Maria and all the other campaign team members:

Speaking dates for Pennsylvania:

7/10/98 Harrisburg
7/15/98 Hershey
7/18/98 Philadelphia
7/23/98 Pittsburgh

Once you have done this, wait for any email responses from the team members, and follow any directions they give you. I'll stop the test in about 45 minutes[13]. Don't forget to "think aloud" as much as you can.

G. Debriefing questionnaire

Questionnaire to follow part one of PGP Usability Test

1. On a scale of 1 to 5, how important did you think the security was in this particular test scenario, where 1 is least important and 5 is most important? 1 2 3 4 5

2. If you generated a key pair during this portion of the test, was there any particular reasoning behind your choice of key type and key size? If so, what was it?

3. Was there anything you thought about doing but then decided not to bother with?

4. Is there anything you think you would have done differently if this had been a real scenario rather than a test?

5. Were there any aspects of the software that you found particularly helpful?

6. Were there any aspects of the software that you found particularly confusing?

7. Are there any other comments you'd like to make at this time?

[13] Our initial plan was to conduct the test in two 45 minute parts, but in practice it turned out to work better not to stop in the middle. After the first couple of sessions the test monitor started telling them that although this document said 45 minutes the test monitor would probably have them just continue for the full 90 minutes.