Why Johnny Can't Encrypt:
A Usability Evaluation of PGP 5.0

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

J. D. Tygar[1]
EECS and SIMS
University of California
Berkeley, CA 94720
tygar@cs.berkeley.edu

Abstract

User errors cause or contribute to most computer security failures, yet user interfaces for security still tend to be clumsy, confusing, or near-nonexistent. Is this simply due to a failure to apply standard user interface design techniques to security? We argue that, on the contrary, effective security requires a different usability standard, and that it will not be achieved through the user interface design techniques appropriate to other types of consumer software.

To test this hypothesis, we performed a case study of a security program which does have a good user interface by general standards: PGP 5.0. Our case study used a cognitive walkthrough analysis together with a laboratory user test to evaluate whether PGP 5.0 can be successfully used by cryptography novices to achieve effective electronic mail security. The analysis found a number of user interface design flaws that may contribute to security failures, and the user test demonstrated that when our test participants were given 90 minutes in which to sign and encrypt a message using PGP 5.0, the majority of them were unable to do so successfully.

We conclude that PGP 5.0 is not usable enough to provide effective security for most computer users, despite its attractive graphical user interface, supporting our hypothesis that user interface design for effective security remains an open problem. We close with a brief description of our continuing work on the development and application of user interface design principles and techniques for security.

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 [2] 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 [4, 9].

This is inescapably a user interface design problem. Legal remedies, increased automation, and user training provide only limited solutions. Individual users may not have the resources to pursue an attacker legally, and may not even realize that an attack took place. Automation may work for securing a communications channel, but not for setting access control policy when a user wants to share some files and not others. Employees can be required to attend training sessions, but home computer users cannot.

Why, then, is there such a lack of good user interface design for security? Are existing general user interface design principles adequate for security? To answer these questions, we must first understand what kind of usability security requires in order to be

[1] Also at Computer Science Department, Carnegie Mellon University (on leave).

effective. In this paper, we offer a specific definition of usability for security, and identify several significant properties of security as a problem domain for user interface design. The design priorities required to achieve usable security, and the challenges posed by the properties we discuss, are significantly different from those of general consumer software. We therefore suspect that making security usable will require the development of domain-specific user interface design principles and techniques.

To investigate further, we looked to existing software to find a program that was representative of the best current user interface design for security, an exemplar of general user interface design as applied to security software. By performing a detailed case study of the usability of such a program, focusing on the impact of usability issues on the effectiveness of the security the program provides, we were able to get valuable results on several fronts. First, our case study serves as a test of our hypothesis that user interface design standards appropriate for general consumer software are not sufficient for security. Second, good usability evaluation for security is itself something of an open problem, and our case study discusses and demonstrates the evaluation techniques that we found to be most appropriate. Third, our case study provides real data on which to base our priorities and insights for research into better user interface design solutions, both for the specific program in question and for the domain of security in general.

We chose PGP 5.0[2] [5, 14] as the best candidate subject for our case study. Its user interface appears to be reasonably well designed by general consumer software standards, and its marketing literature [13] indicates that effort was put into the design, stating that the "significantly improved graphical user interface makes complex mathematical cryptography accessible for novice computer users." Furthermore, since public key management is an important component of many security systems being proposed and developed today, the problem of how to make the functionality in PGP usable enough to be effective is widely relevant.

We began by deriving a specific usability standard for PGP from our general usability standard for security. In evaluating PGP 5.0's usability against that standard, we chose to employ two separate evaluation methods: a direct analysis technique called cognitive walkthrough [17], and a laboratory user test [15]. The two methods have complementary strengths and weaknesses. User testing produces more objective results, but is necessarily limited in scope; direct analysis can consider a wider range of possibilities and factors, but is inherently subjective. The sum of the two methods produces a more exhaustive evaluation than either could alone.

We present a point by point discussion of the results of our direct analysis, followed by a brief description of our user test's purpose, design, and participants, and then a compact discussion of the user test results. A more detailed presentation of this material, including user test transcript summaries, may be found in [18].

Based on the results of our evaluation, we conclude that PGP 5.0's user interface does not come even reasonably close to achieving our usability standard – it does not make public key encryption of electronic mail manageable for average computer users. This, along with much of the detail from our evaluation results, supports our hypothesis that security-specific user interface design principles and techniques are needed. In our continuing work, we are using our usability standard for security, the observations made in our direct analysis, and the detailed findings from our user test as a basis from which to develop and apply appropriate design principles and techniques.

2 Understanding the problem

2.1 Defining usability for security

Usability necessarily has different meanings in different contexts. For some, efficiency may be a priority, for others, learnability, for still others, flexibility. In a security context, our priorities must be whatever is needed in order for the security to be used effectively. We capture that set of priorities in the definition below.

Definition: 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.**

[2] At the time of this writing, PGP 6.0 has recently been released. Some 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. Also, our evaluation was performed using the Apple Macintosh version, but the user interface issues we address are not specific to a particular operating system and are equally applicable to UNIX or Windows security software.

2.2 Problematic properties of security

Security has some inherent properties that make it a difficult problem domain for user interface design. Design strategies for creating usable security will need to take these properties explicitly into account, and generalized user interface design does not do so. We describe five such properties here; it is possible that there are others that we have not yet identified.

1. 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. Furthermore, if security is too difficult or annoying, users may give up on it altogether.

2. 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.

3. 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.

4. 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.

5. 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 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.

2.3 A usability standard for PGP

People who use email to communicate over the Internet need 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 general definition of usability for security, we derived the following question on which to focus our evaluation:

> *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?*

Stating 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
- 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.

3 Evaluation methods

We chose to evaluate PGP's usability through two methods: an informal cognitive walkthrough [17] 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 2.3; and a user test [15] 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 [7] 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 [11]. 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 2.1 and its restatement specifically for PGP in Section 2.3. 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. 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 [19], we have found little existing material on user testing that addresses similar concerns.

4 Cognitive walkthrough

Since this paper is intended for a security audience, and is subject to space limitations, we present the results of our cognitive walkthrough in summary form, focusing on the points which are most relevant to security risks.

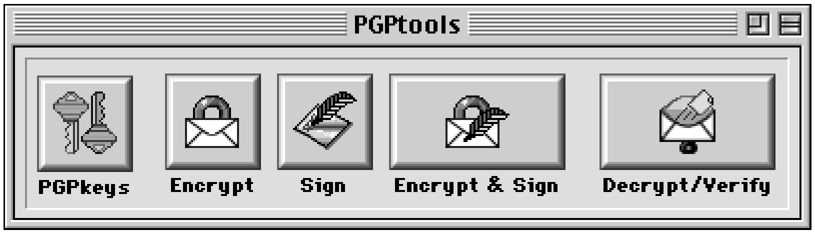

Figure 1

4.1 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 1, 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 appear to be straightforward visual metaphors that help make the use of keys to encrypt and decrypt into an intuitive concept.

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.

4.2 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 2. Second, when users attempt to encrypt documents using mixed key types, a warning message

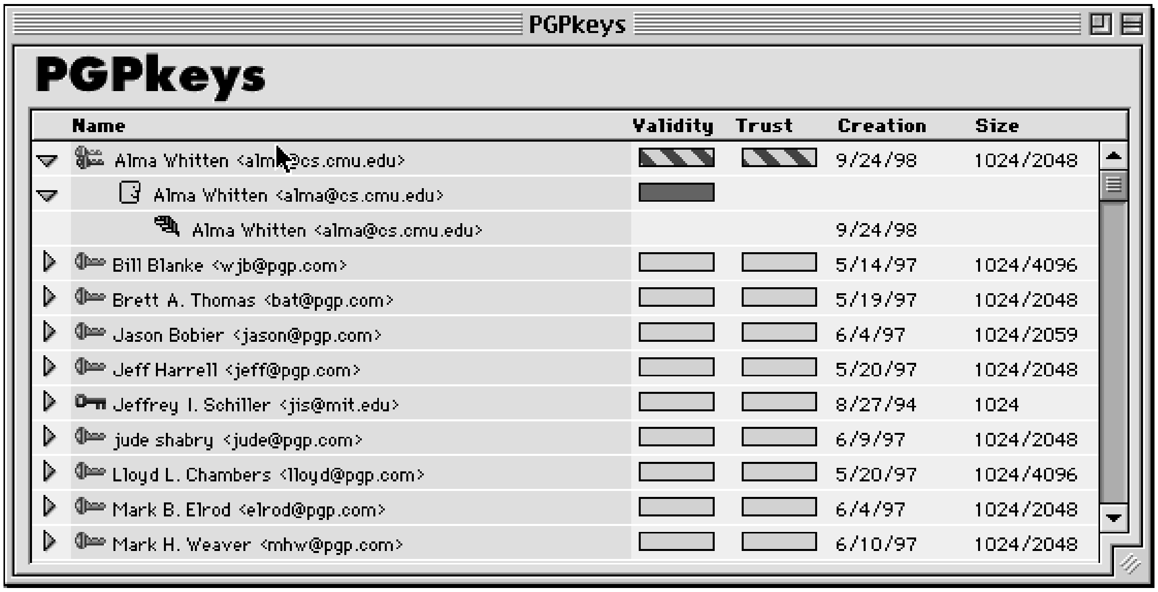

Figure 2

is displayed to tell them that recipients who have earlier versions of PGP may not be able to decrypt it.

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 3) 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.

4.3 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 shown in Figure 4: 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.

At present, PGPKeys keeps no records of key server accesses. There is nothing to show whether a

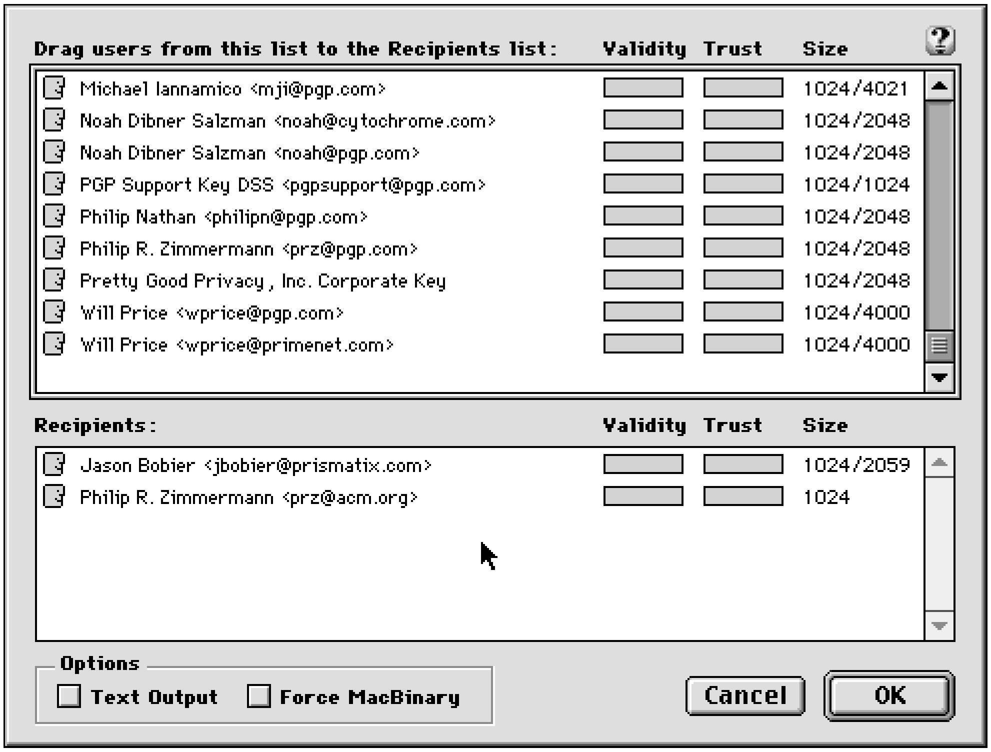

Figure 3

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 to the user for key management and for verifying that key server operations were completed 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.

4.4 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.

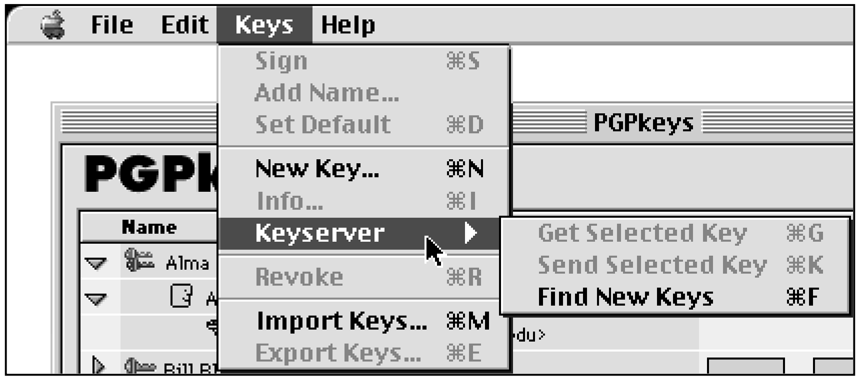

Figure 4

4.5 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 able to be decrypted, 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 above, 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.

4.6 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.

4.7 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 2) 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. Some security experts may find the downplaying of this information alarming, but the goal here is to enable users who are inexperienced with cryptography to understand and begin to use the basics, and to prevent confusion or frustration that might lead them to use PGP incorrectly or not at all.

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, for the reasons discussed in Section 4.4.

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 User test

5.1 Purpose

Our user test was designed to evaluate whether PGP 5.0 meets the specific usability standard described in Section 2.3. 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.

5.2 Description

5.2.1 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.

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 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[3].

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

[3] This aspect of the test may trouble the reader 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.

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 decrypt 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.

5.2.2 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. Their ages ranged from 20 to 49, and their professions were diversely distributed, including graphic artists, programmers, a medical student, administrators and a writer. More detailed information about participant selection and demographics is available in [18].

5.3 Results

We summarize the most significant results we observed from the test sessions, again focusing on the usability standard for PGP that we gave in Section 2.3. Detailed transcripts of the test sessions are available in [18].

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[4] 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

[4] 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.

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 and P10 eventually succeeded in sending correctly encrypted email to the team members before the end of the 90 minute test session (P9 figured out that she needed to use the campaign manager's public key, but then sent email to the the entire team encrypted only with that key), 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.

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 four who did not succeed, 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 ultimately failed.

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 a reply email from the test monitor posing as the team member with an RSA key, 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.

6 Conclusions

6.1 Failure of standard interface design

The results seen in our case study support our hypothesis that the standard model of user interface design, represented here by PGP 5.0, is not sufficient to make computer security usable for people who are not already knowledgeable in that area. Our twelve test participants were generally educated and experienced at using email, yet only one third of them were able to use PGP 5.0 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.

In Section 2.1, we defined usability for security in terms of four necessary qualities, which translate directly to design priorities. PGP 5.0's user interface fails to enable effective security where it is not designed in accordance with those priorities: test participants did not understand the public key model well enough to know that they must get public keys for people they wish to send secure email to; many who knew that they needed to get a key or to encrypt still had substantial difficulties in figuring out how to do so; some erroneously sent secrets in plaintext, thinking that they had encrypted; and many expressed frustration and unhappiness with the experience of trying to use PGP 5.0, to the point where it is unlikely that they would have continued to use it in the real world.

All this failure is despite the fact that PGP 5.0 is attractive, with basic operations neatly represented by buttons with labels and icons, and pull-down menus for the rest, and despite the fact that it is simple to use for those who already understand the basic models of public key cryptography and digital signature-based trust. Designing security that is usable enough to be effective for those who don't already understand it must thus require something more.

6.2 Usability evaluation for security

Since usable security requires user interface design priorities that are not the same as those of general consumer software, it likewise requires usability evaluation methods that are appropriate to testing whether those priorities have been sufficiently achieved. Standard usability evaluation methods, simplistically applied, may treat security functions as if they were primary rather than secondary goals for the user, leading to faulty conclusions. A body of public work on usability evaluation in a security context would be extremely valuable, and will almost certainly have to come from research sources, since software developers are not eager to make public the usability flaws they find in their own products.

In our own work, which has focused on personal computer users who have little initial understanding of security, we have assigned a high value to learnability, and thus have found cognitive walkthrough to be a natural evaluation technique. Other techniques may be more appropriate for corporate or military users, but are likely to need similar adaptation to the priorities appropriate for security. In designing appropriate user tests, it may be valuable to look to other fields in which there is an established liability for consumer safety; such fields are more likely to have a body of research on how best to establish whether product designs successfully promote safe modes of use.

6.3 Toward better design strategies

The detailed findings in our case study suggest several design strategies for more usable security, which we are pursuing in our ongoing work. To begin with, it is clear that there is a need to communicate an accurate conceptual model of the security to the user as quickly as possible. The smaller and simpler that conceptual model is, the more plausible it will be that we can succeed in doing so. We thus are investigating pragmatic ways of paring down security functionality to that which is truly necessary and appropriate to the needs of a given demographic, without sacrificing the integrity of the security offered to the user.

After a minimal yet valid conceptual model of the security has been established, it must be communicated to the user, more quickly and effectively than has been necessary for conceptual models of other types of software. We are investigating several strategies for accomplishing this, including the possibility of carefully crafting interface metaphors to match security functionality at a more demanding level of accuracy.

In addition, we are looking to current research in educational software for ideas on how best to guide users through learning to manage their security. We do not believe that home users can be made to cooperate with extensive tutorials, but we are investigating gentler methods for providing users with the right guidance at the right time, including how best to make use of warning messages, wizards, and other interactive tools.

7 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 [12, 20], 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 [10], 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.

Ross Anderson discusses the effects of user non-compliance on security in [1], and Don Davis analyzes the unrealistic expectations that public-key based security systems often place on users in [3].

Beyond that, we know only of one paper on usability testing of a database authentication routine [8], and some brief discussion of the security and privacy issues inherent in computer supported collaborative work [16]. John Howard's thesis [6] provides interesting analyses of the security incidents reported to CERT[5] between 1989 and 1995, but focuses more on

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

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.

Acknowledgements

We thank Robert Kraut for helpful advice on the design of our user test. This publication was supported in part 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.

References

1. Ross Anderson. Why Cryptosystems Fail. In *Communications of the ACM*, 37(11), 1994.

2. Matt Bishop. *UNIX Security: Threats and Solutions*. Presentation to SHARE 86.0, March 1996. At `http://seclab.cs.ucdavis.edu/~bishop/scriv/1996-share86.pdf.`

3. Don Davis. Compliance Defects in Public-Key Cryptography. In *Proceedings of the 6th USENIX Security Symposium*, 1996.

4. The Economist. *The End of Privacy*. May 1, 1999, pages 21-23.

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

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

7. 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).

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

9. 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.

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

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

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

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

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

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

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

17. 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.

18. Alma Whitten and J.D. Tygar. *Usability of Security: A Case Study*. Carnegie Mellon University School of Computer Science Technical Report CMU-CS-98-155, December 1998. `ftp://reports-archive.adm.cs.cmu.edu/1998/CMU-CS-98-155.ps`

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

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