

# Making Security Usable

Alma Whitten

May 2004

CMU-CS-04-135

School of Computer Science  
Carnegie Mellon University  
5000 Forbes Avenue  
Pittsburgh, PA 15213-3890

Thesis Committee:  
J. Douglas Tygar, Chair  
Robert Kraut  
Steven Roth  
Edward Felten, Princeton University

*Submitted in partial fulfillment of the requirements for the degree of  
Doctor of Philosophy.*

© 2004 by Alma Whitten

**Keywords:** security, privacy, usability, cryptography, user interfaces, learnability, electronic mail, conceptual models, visual metaphors, scaffolding, metaphor tailoring, safe staging.

## Abstract

Usability remains one of the most pressing and challenging problems for computer security. Despite widespread recognition of the damage that results from configuration errors and other user misunderstandings, little progress has been made toward making verifiably usable security a reality. In this dissertation, we propose that the usability problem for security is difficult to solve precisely because security presents qualitatively different types of usability challenges from those of other types of software, and suggest that making security usable will require the creation of user interface design methods that address those challenges.

We begin by presenting an analysis of security as a usability domain. Our analysis is founded on the identification of five characteristics of computer security which distinguish the problem of creating usable security from the general problem of creating usable consumer software. Working from those characteristics, we establish a set of guidelines for determining when security can safely be automated and hidden from the user versus when it must be made visible and usable. We argue for a design philosophy that considers the benefits of presenting a security mechanism as a general purpose tool rather than an application specific appliance, and discuss some of the additional ethical and pragmatic issues raised by questions of visibility. Our analysis concludes with the identification of a design principle for usable security, *well-in-advance*, that stands in contrast to the general user interface design practice of providing just-in-time information.

In order to ground our analysis with some empirical data, we describe a case study that we conducted using PGP 5.0 for the Apple Macintosh. PGP is a commercial product for adding cryptographic security to electronic mail. Marketing literature for PGP 5.0 claimed that its user interface had been substantially improved and was highly usable, and it was often mentioned in the security community as an example of a good user interface for a security program. We agreed that the PGP 5.0 user interface appeared good by conventional standards, but suspected that it would fail to meet the special challenges posed by security. A cognitive walkthrough analysis and an extensive user test demonstrated that this was indeed the case.

We next present two specialized user interface design techniques that we have developed. *Safe staging* is founded on the well-in-advance principle, and combines the concept of staged user interfaces with a safety template derived from established standards for consumer product warning labels, resulting in a technique for designing user interfaces that allow users to safely and consciously postpone learning about the use of particular security mechanisms until they decide they are ready to do so. *Metaphor tailoring* uses conceptual model specifications that have been augmented with security risk information to create visual representations of security mechanisms and data that incorporate as many desirable visual cues as possible. We demonstrate the use of both these techniques by applying them to the design of a user interface for a hypothetical secure electronic mail program called *Lime*.

To evaluate the success of our techniques as represented in our user interface design, we first performed comparative user tests using paper presentations. Two versions of these tests were conducted, each using three presentation variants. The first version of the test was designed to compare a design that used safe staging to present the mechanism of key certification against two other variants that did not use staging. This version yielded a strong positive result for staging: 45% participant success versus 10% and 0% for the unstaged variants. The second version of the test was designed to compare two variants of the tailored metaphors used to present public key cryptography in Lime against the standard images used in PGP 5.0. This test was judged to be a failure due to a problem in the test design, and did not yield useful results.

For the next part of our evaluation, we created a complete working software implementation of the Lime design, with simulated cryptographic and electronic mail functionality, and used it to conduct an extensive user test. Results from this test were good: not only were nearly all participants able to use the basic cryptographic functions successfully and appropriately, but most were also able to make appropriate use of key certification, which is one of the most difficult cryptographic concepts to make usable. The combined results of the first comparative test and the software user test support our thesis that it is possible to make security usable for general consumer software, when user interface design techniques developed for the specific needs of security are used.

*This dissertation is dedicated to my father, Franklin Cacciutto.*

## Acknowledgements

First and foremost, a multitude of thanks are due to my thesis advisor, Doug Tygar, for his support, patience, insight, creativity and open-mindedness during the long course of my thesis work. I have been fortunate to have the support and mentoring of exceptional advisors throughout my academic career, and would also like to thank Joseph Bates, who was my original advisor at CMU, and Robert McCartney and Neal Alderman, who guided and encouraged me at the University of Connecticut.

I am especially grateful to Doug for his guidance in choosing my thesis committee. Steve Roth, Bob Kraut and Ed Felten all provided generous encouragement and invaluable feedback, and they have my deepest thanks. Heartfelt thanks also go to Sharon Burks for all of her advocacy, sensible advice and warm support, which made a tremendous difference to me at many points in my graduate school career.

My fellow security graduate students were a wonderful resource for me, both at CMU and at Berkeley. Howard Gobioff, Michael Harkavy, Jean Camp, Bennet Yee, Adrian Perrig, Rachna Dhamija, Dawn Song, Monica Chew and Mahad Ibrahim: thank you for your encouragement and friendship. Thank you also to Barbara Goto, whose competence and good cheer should be an inspiration to us all. And special thanks are due to Ka-Ping Yee for his assistance as a second coder for my user study results – thanks Ping!

Working with the Postage Technology Management group from USPS has been one of the most enjoyable aspects of my time in graduate school. I'd like to thank all of them for their support and for being such a pleasure to work with, but most especially Wayne Wilkerson, Roy Gordon, and Dan Lord.

I have also been fortunate to have the unstinting support and encouragement of my colleagues at Google throughout my final thesis revisions. Thanks to Fritz Schneider, Bill Coughran, Niels Provos, Jun Luo, Bryan Olson, and once again, Howard Gobioff and Bennet Yee, all of whom repeatedly reassured me that finishing my thesis was a perfectly valid use of Google 20% time.

Thanks also to the many other graduate students and members of the CMU SCS community who made my time there memorable: Joseph O'Sullivan, Paul Placeway, Daniel Tunkelang, Prem Janardhan, Morgan Price, Rowan Davies, Ralph Melton, Corey Kosak, Nick Thompson, and above all Seán Slattery. And thanks to Karl Wurst and Amy West, without whom I would not have made it to CMU in the first place.

Special thanks to my wonderful in-laws, Karen and David Willmott, for sheltering, encouraging, and insisting on feeding me while I wrote the last chapters of my thesis draft; this document should definitely be stamped "Made in New Zealand."

My father, Franklin Cacciutto, earned the first doctorate in our family in 1991, and raised me to believe that the only things a little ingenuity can't fix are the crack of dawn and a broken heart. This dissertation is dedicated to him.

Finally and always, my thanks and love to Andrew Willmott, whose unwavering support and faith in me makes all things possible.

# Contents

|   |     |
|---|-----|
| CHAPTER 1 Introduction .....                            | 1   |
| 1.1 The need for usable computer security .....         | 1   |
| 1.2 Overview of research on usable security .....       | 2   |
| 1.3 Summary of our contributions .....                  | 3   |
| 1.4 Contents of dissertation .....                      | 5   |
| CHAPTER 2 Analysis, principles and guidelines .....     | 6   |
| 2.1 Security compared to other types of software.....   | 6   |
| 2.2 Properties that make security difficult.....        | 7   |
| 2.3 Making security invisible.....                      | 8   |
| 2.4 Generality: tools, appliances and hierarchies.....  | 11  |
| 2.5 The well-in-advance principle.....                  | 12  |
| 2.6 Summary .....                                       | 13  |
| CHAPTER 3 Case study: PGP 5.0 .....                     | 14  |
| 3.1 Case study goals .....                              | 14  |
| 3.2 Methodology .....                                   | 15  |
| 3.3 Results.....  | 18  |
| 3.4 Conclusions.....                                    | 21  |
| CHAPTER 4 Design technique: safe staging .....          | 23  |
| 4.1 A brief history of staged user interfaces .....     | 23  |
| 4.2 Adapting staging for computer security .....        | 24  |
| 4.3 The design technique .....                          | 25  |
| 4.4 Example: staging applet security.....               | 27  |
| 4.5 Variant: staging for the abstraction property ..... | 29  |
| 4.6 Safe staging demonstrated in Lime .....             | 30  |
| CHAPTER 5 Design technique: metaphor tailoring.....     | 40  |
| 5.1 Risk enumeration.....                               | 40  |
| 5.2 Metaphor tailoring.....                             | 43  |
| 5.3 Metaphor tailoring demonstrated in Lime .....       | 45  |
| 5.4 Additional example: firewall port management.....   | 52  |
| 5.5 Summary .....                                       | 53  |
| CHAPTER 6 Experimental results .....                    | 54  |
| 6.1 Staged user interface comparison.....               | 55  |
| 6.2 Visual metaphor set comparison .....                | 59  |
| 6.3 Proof of concept user testing.....                  | 63  |
| 6.4 Evaluation summary .....                            | 79  |
| CHAPTER 7 Conclusions and future work.....              | 81  |
| 7.1 Contributions.....                                  | 81  |
| 7.2 Future work.....                                    | 82  |
| 7.3 Concluding remarks.....                             | 84  |
| APPENDIX A PGP case study materials and data.....       | 85  |
| A.1 Description of test participants.....               | 85  |
| A.2 Description of testing process .....                | 87  |
| A.3 Summaries of test session transcripts .....         | 90  |
| A.4 Test materials .....                                | 103 |



|  |     |
|--|-----|
| APPENDIX B Staging comparison test materials and data .....  | 106 |
| B.1 Participant screening questions .....                    | 107 |
| B.2 Briefing given to participants .....                     | 108 |
| B.3 Presentation variants .....                              | 108 |
| B.4 Questions and collated results .....                     | 116 |
| APPENDIX C Metaphor comparison test materials and data ..... | 156 |
| C.1 Presentation variants .....                              | 156 |
| C.2 Questions and collated results .....                     | 166 |
| APPENDIX D Lime user test materials and data .....           | 204 |
| D.1 Participant intake questionnaire .....                   | 205 |
| D.2 Participant initial briefing materials .....             | 206 |
| D.3 Scenario descriptions given to participant .....         | 207 |
| D.4 Post-test questionnaire .....                            | 211 |
| D.5 Collated results for true/false questions .....          | 214 |
| References .....   | 216 |

## **CHAPTER 1 Introduction**

### **1.1 The need for usable computer security**

Most computer security is not easy for people to use. Even before the Internet became a household word, security failures were more likely to be caused by user errors than by weak cryptography or bad protocol design. This was true even when networked computers were used primarily by people who had some degree of technical skill or professional training; it is overwhelmingly more true now that a personal computer with an Internet connection has become a standard consumer good. Hundreds of millions of people now use the Internet to communicate, find information, and conduct financial transactions on a regular basis. Ideally, they should be empowered to make and enforce their own security and privacy decisions, but the usability barrier has so far made this implausible.

This is all the more frustrating because a wealth of knowledge already exists on how to make software usable. Literature on human-computer interaction, user-centered design, and usability engineering is readily available and widely used in software development. Web browsers, electronic mail, instant messaging, and many other software applications are used daily and successfully by people who have very little technical background. But when similar user interfaces have been created for security, they have had little or no success.

There are several possible explanations for this. Perhaps the designers simply did not do a good job of applying standard usability design techniques. Maybe most people just are not interested in security, and do not want to use it, regardless of how well it is presented. Possibly the security in question is inherently too difficult and can never be made usable for non-experts. Each of these explanations may be true to some degree for some cases. Another possible explanation, which we explore in this dissertation, is this:

**Many crucial usability problems in computer security are fundamentally different from those in most other consumer software, and usability design techniques need to be carefully adapted and prioritized in order to solve them successfully.**

By examining the ways in which security goals and tasks differ from those of most consumer software applications, we identify common problems that are not well addressed by standard usability design techniques<sup>1</sup>. This enables us to develop and evaluate modified and enhanced techniques that are much more effective for designing usable security. These techniques may not be equally applicable to all aspects of computer security; indeed, as we acknowledged earlier, some security may be inherently unusable, and some security may already be usable enough, at least for particular sets of users in particular concepts. However, we argue that they are broadly applicable to many of the most pressing usability problems in security for personal computing, and thus are a significant contribution toward the general problem of making security usable.

## 1.2 Overview of research on usable security

Up until about five years ago, there was little research directly investigating usability for security. [Anderson94] helped to provoke awareness that security tends to fail where it interacts with the messiness of the real world, rather than because of weaknesses in the underlying protocols. A few years later, [Davis96] provided an analysis of the ways in which public key cryptography was often deployed with extremely unrealistic expectations about how the people in the system could be expected to behave, and [Zurko96] began calling for a user-centered design approach to computer security. Soon after that, we conducted the study that is discussed in Chapter 3 of this dissertation [Whitten98, Whitten99], finding that even though a well-known security program had been given what appeared to be a very nice user interface, it remained unusable for people who did not have substantial prior knowledge of the security technology involved.

Over the next few years, interest in usability for security grew steadily, to the point where mailing lists and workshops devoted entirely to that topic have come into being. Much interesting work has been done during that time, taking a variety of approaches to the problem, most of which are dissimilar but complementary to our own. These approaches might be categorized as follows:

- Ethnographic or similar investigation into how people currently think about security and privacy, both in terms of how they make sense of the technology and in terms of what concerns them most. [Adams99, Adams99-2, Adams00, Dourish03, Friedman02, Friedman02-2, and Karvonen99] have taken this approach. Dourish in particular has argued that computer security must be redefined and redeployed to better match the actual concerns of real people; we completely agree. After that has been done, however, the problem of how to

---

<sup>1</sup> One might ask whether the failing is not in the technique, but in the designer who fails to apply it with enough insight and creativity. After all, it is standard in usability engineering to carefully consider the goals and tasks that the user will need to perform. Our response is that if a few designs fail, perhaps those products need better designers, but if all designs fail, then the designers probably need better techniques.

present the necessary technology will remain, and our work contributes techniques and guidelines for solving that problem effectively.

- Investigation of ways to greatly reduce the need for the user to understand security, either by making the security entirely invisible and automated, or by making the choices available to the user very coarse-grained and basic. Examples of this approach include [Brown99 and Garfinkel03, Garfinkel03-2] who propose to automate key distribution and management for email security, and [Balfanz00], who reduces desktop computer security management to a choice between several predefined virtual desktops with different security policies, leveraging the fact that people have a good understanding of security through physical separation. Again, we consider these approaches valuable both for reducing the user interface design problem to be solved, and for raising the default level of security available. However, we consider them to be fundamentally limited and also not always in the best interests of the user, for reasons that we discuss in Chapter 2.
- Analyses of the usability needs of security from a standard usability engineering perspective [Adams99-3, Dufft99, Laukka00, Shen92, Wolf99 and Zurko96], from the perspective argued in our earlier work [Holmström99, Karvonen99], and from a security perspective [Yee03].
- Experiments in designing improved user interfaces for particular security problems, including the design of security and privacy wizards [Cranor99, Damker99, Friedman02-3, Jendricke00, Lau99, Ye02, Zurko99].
- Research into new security technologies that are inherently easier to use, such as biometric authentication and various kinds of graphical passwords, such as [Brostoff00, Coventry02, Dhamija00, Dhamija00-2 and Jermyn99].
- Research into how to evaluate the usability of security applications [Ammenwerth99 and Karat89].

## 1.3 Summary of our contributions

This dissertation is based on the thesis that many of the usability requirements of computer security are fundamentally different from those of general end-user software, and that analysis of those requirements can lead to new design principles and techniques that are effective for creating usable security software when standard techniques fail. The contributions of the research include the following:

Concepts and techniques:

- An analysis of the particular usability requirements of computer security.
- A set of usability design principles particular to computer security.

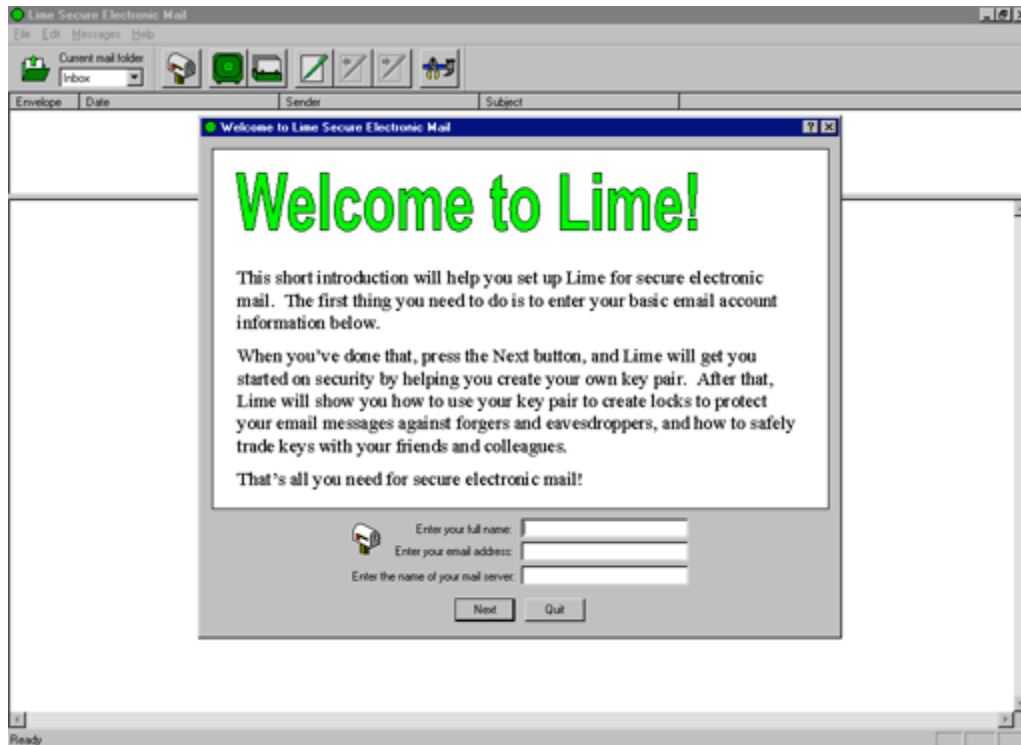


Figure 1-1: Opening screen of Lime (50% scale)

- A technique called safe staging that allows the user to safely postpone learning how to use a security mechanism.
- A technique for designing effective visual representations of security mechanisms.

#### Artifacts:

- Lime (see Figure 1-1), which is a working implementation of a user interface design for usable public key cryptography-based secure electronic mail software, coded in Visual C++ and executable in most versions of Microsoft Windows.
- Limeserver, a combination mail and key server simulator, coded in C++, executable in UNIX and intended for use in conducting user tests with Lime.
- A hierarchical design for the visual representation of public key cryptography operations, which may be further specialized and extended for use in a wide variety of security applications.

#### Experimental results:

- A case study, with formal user testing, of the usability of a commercial security software product which claimed to be highly usable, showing that despite its

conventionally friendly graphical user interface, most test participants were unable to use it successfully for basic security tasks.

- A paper-based user test of the effectiveness of presenting public key cryptography using safe staging for the mechanism of key certification, compared to that of two variant presentations that did not use staging, showing that participants who were given the staged variant performed significantly better at describing how key certification would be used.
- An extensive formal user test of Lime, showing that almost all participants were able to successfully use it for most security tasks, and that most participants were able to successfully use it for advanced security tasks.

## 1.4 Contents of dissertation

We begin in the next chapter with an analysis of what it must mean for security to be usable, and of how the most important usability problems in security differ from those in more standard varieties of consumer software. Chapter 3 then presents our preliminary case study of PGP 5.0 for the Apple Macintosh, chosen because it was advertised and frequently cited as having an excellent user interface. Our testing showed that the majority of our test participants were unable to use PGP to correctly sign and encrypt an email message, despite the fact that they were all experienced email users.

In the second part of this dissertation, we present two usability design techniques that are specialized for computer security, and demonstrate them with a user interface design for a secure email program called *Lime*. Chapter 4 introduces *safe staging*, which takes the basic concept of multi-level user interfaces, which are usually designed to aid learning and to support both novice and expert users, and enhances it by providing a clear theory of how to design levels and transitions that preserve the user's security at all times. Chapter 5 introduces *metaphor tailoring*, which begins with existing techniques for designing visual user interface metaphors and adds a new technique, *risk enumeration*, that enables us to tailor our visual representation to the most important aspects of the security in a methodical and prioritized way.

Chapter 6 presents the results of several user tests of Lime. Compared against existing ways of presenting public key cryptography, Lime's safe staging presentation yielded significantly better results for user understanding of key signing. Proof of concept testing with a Windows implementation of Lime showed that most participants were able to successfully sign and encrypt email, and many were also able to use key signing to securely identify the owners of cryptographic keys, despite having no knowledge of public key cryptography prior to the test. Finally, Chapter 7 summarizes the contributions and conclusions of this thesis.

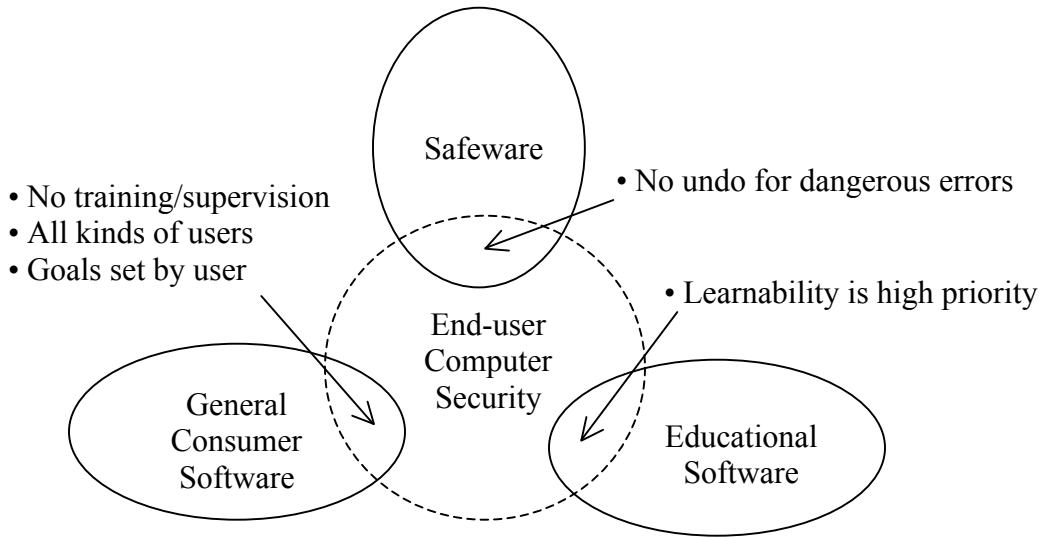
## CHAPTER 2 Analysis, principles and guidelines

In this chapter, we present an analysis of the similarities and differences between computer security and other types of software from a usability perspective. We identify five special properties of computer security that make usability difficult, which will form the starting motivation for much of our work. We address the fundamental question of whether the usability problems in computer security can be resolved by making security invisible and automated, and present a set of rules for determining whether invisible security is a valid approach for a particular software application. Exploring the question of visibility further, we then discuss a number of ethical and pragmatic considerations to be taken into account when deciding how to present a particular security mechanism to users. Finally, we argue for a security-specific design principle, *well-in-advance*, which will be important in later chapters.

### 2.1 Security compared to other types of software

End-user computer security has qualities in common with general consumer software, with safeware such as the software used to control medical systems and airplanes [Leveson95], and with educational software. A graphical representation of the overlapping sets of usability challenges for these different types of software is shown in Figure 2-1.

As with safeware, computer security users must avoid making a variety of dangerous errors, because once those errors are made, it is difficult or impossible to reverse their effects. Safeware, however, may be expected to be used by a preselected and trained group, to achieve goals defined by experts; none of those expectations hold for end-user computer security, which in those respects is more like general consumer software.



**Figure 2-1: Security in relation to other types of software**

Finally, because computer security badly needs to be learnable, we may attempt to draw on techniques developed for educational software, but we will be constrained by the limits of user willingness to cooperate with tutorials for the sake of safety rather than as a means to accomplishing their primary goals.

## 2.2 Properties that make security difficult

### **The secondary goal 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.



### **The hidden failure 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.

### **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 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 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 Making security invisible**

One fatalistic view among computer security professionals holds that the only way to make security truly reliable is to make it completely automated and invisible to users, thereby removing user error as a source of security failures. Unfortunately, in practice, making a particular component of the security invisible will often lead to a different set of security risks, equally as serious as any that were prevented. These new risks are of two types: risks that a user will be led to take actions based on inaccurate expectations

### **Rules for making security invisible**

1. If a user action in the application depends on a particular security function for protection, and there is any possibility that the security function may sometimes not be able to be executed, then, in the case that the security function cannot be executed, one of the following clauses **MUST** be met:
  - a. The user action **MUST** be completely disallowed, both inside and outside the application.
  - b. Or, the lack of protection for the user action **MUST** be made visible to the user, and tools for remedying the problem that prevents the execution of the security function **SHOULD** be made available to the user.
2. If a security policy in the application determines who is granted access to resources that the user owns, then both of the following clauses apply:
  - a. That security policy **MUST** be made visible to the user.
  - b. Tools for modifying that security policy **SHOULD** be made visible to the user.

**Figure 2-2: Invisibility rules**

about whether they will be invisibly protected while doing so, and risks that a user will be regrettably pushed toward taking unprotected actions in the event that the only presented alternative is to refrain from the actions altogether.

Figure 2-2 presents a set of rules which can be used to determine whether a particular component of the security for a specific software application, intended to be run in a particular setting, can be safely made invisible. Note that, since the first type of risk described involves the software failing to be honest toward its users, the rules that relate to that risk use the word “must”, while those relating to the second type of risk are less ethically imperative and thus use the word “should”.

In practice, the first rule requires that if a security function that protects a particular user action is to be made invisible, and if that function depends on the availability of any additional data, such as keys or policies, then the software application in question must be one which runs strictly within a closed environment such as a set of corporate offices, so that an expert support staff will be present to attend to security configuration and maintenance for all the computers in the system, and also so that users can be restricted from using alternative software programs to attempt disallowed actions. Otherwise, user

actions will regularly be disallowed, and users will, quite understandably, turn to some more accommodating software program to accomplish their goals. Furthermore, the invisibly secured software program should not be designed to resemble any non-secure software program intended for more general use, lest it lead its users to expect invisible protection in other circumstances where none exists.

As for the second rule, it requires that security policies that control access to resources must always be visible to the owners of those resources, so that they are able to make informed decisions about whether to keep the resources in the system at all. This does not mean that such policies can never be made invisible, since in many systems users may be working with corporate resources that they do not own, but it does effectively rule out making access control policies invisible in software intended for home users.

Note that these rules in fact apply to any software that has the potential to create security or privacy risks for its users, and that obedience to the rules need not, in fact, involve any security technology at all, as the rules' fundamental requirement is simply that software provide its users with accurate information for risk assessment.

On the topic of making security visible, several issues remain that ought to be addressed. There are a number of arguments that are commonly made against visibility, with the implication that, despite whatever risks may be created by making security completely invisible, the alternatives are inevitably worse. We will briefly discuss the two such arguments that we have most often seen made, and point out practical and ethical issues that ought to be taken into account before accepting either of them. We will then conclude this section with an argument that, in some cases, it may be worth increasing the visibility of the security in an application even beyond that which is required by the visibility guidelines for that particular application and setting, because doing so may contribute to the usability and general effectiveness of security in a wider context.

As we briefly mentioned earlier, the most commonly made argument against visible security is that users are too ignorant and fallible to be able to protect themselves effectively, and that therefore the only real way to protect users' privacy and security is to design software that does it for them, invisibly and automatically. In addition to the risks delineated by our invisibility guidelines, there are several other reasons to be concerned about this argument. The first such reason is that this argument is self-perpetuating: if security is hidden from users, then users will remain ignorant about security technology, and their continuing ignorance will be used to justify continuing to hide security from them. The second such reason is one of conflict of interest: to argue that users cannot manage their own security is to argue that software manufacturers must manage users' security for them. Those same software manufacturers often have a strong financial interest in collecting data on users' habits, actions and preferences, and in privileging their own software over that of competitors in matters of access control. To put them in control of the very security policies that are intended to guard user privacy and resources is thus to put the fox in charge of the henhouse. These points should be kept in mind when the "ignorant users" argument for invisible security is made by representatives of software manufacturers.

Another, somewhat similar, argument that is often made is that, because security goals are not primary goals for users, software which makes security visible will annoy users, who just want to get on with their primary goals and expect invisible protection while they do so. Again, this is a self-perpetuating argument, in which software manufacturers make security invisible, despite the risks that creates, market their products as protecting user security, and thus generate and support a widespread user expectation that security can be provided invisibly.<sup>2</sup> It is also the case that to date there has been relatively little creative experimentation with ways of presenting security to users; that users have been annoyed by the security demands of existing software does not mean that it is impossible to present security in a way that is attractive and gives users a pleasurable feeling of understanding and control.

A final point, which applies to both of the arguments addressed above, is that in the case of software designs that make security invisible, we need to be wary of assuming success based on a lack of negative feedback. This is true for security software designs in general, but it is particularly true when the security is made invisible, because users who are not seeing their security while it functions will also not see their security fail, and will thus have been removed as a potentially valuable source of feedback about any security failures that occur.

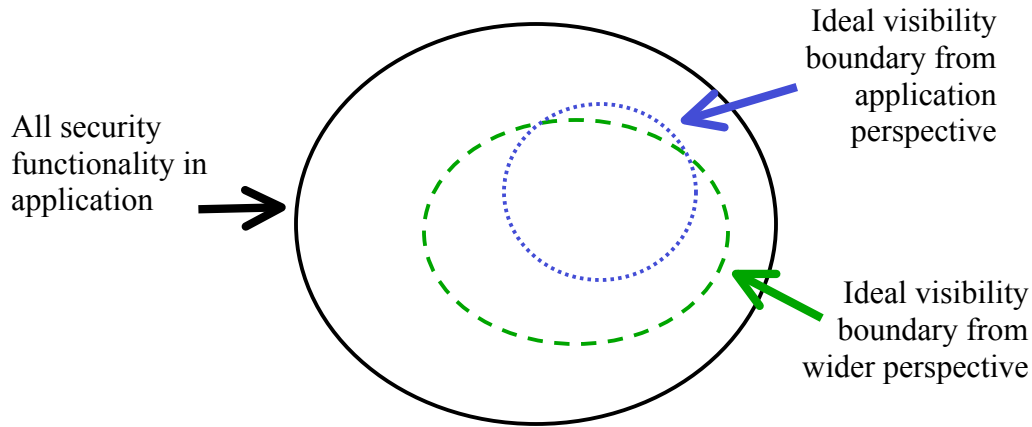
## 2.4 Generality: tools, appliances and hierarchies

The invisibility guidelines presented earlier can be used to determine whether a particular security function must be made visible for a given software program in a given setting. It may often be the case, however, that the desirability of promoting the usability of security in a wider context and over the longer term should also be taken into account, as shown in Figure 2-3, and that doing so will lead us to choose to implement a greater degree of security visibility than is actually required for our particular application and setting.

To see how this might be, it is useful to bring in the concept of *tools* vs. *appliances* [Kolojejchick97]. Briefly, a tool is a low-level, general purpose technology which requires skill to use, but which can be used flexibly and applied usefully in a wide range of situations. An appliance, by contrast, is a high-level, usually single-purpose device, which incorporates its own intelligence about its designated task and thus requires very little skill to use. A paintbrush is a tool; a photocopier is an appliance.

---

<sup>2</sup> This phenomenon was frequently observed during the software user testing that will be described later in this dissertation; when presented with a software program incorporating visible public key cryptography, users often complained during the first 10-15 minutes of the testing that they would expect “that kind of thing” to be handled invisibly. As their exposure to the software continued and their understanding of the security mechanisms grew, they generally ceased to make that complaint.



**Figure 2-3: Different perspectives on visibility for security**

For many security applications, it may well be the case that the locally ideal security interface design, one which meets the visibility guideline requirements and also achieves maximum simplicity and usability, is one which takes a standard, widely used security technology and conceals it within an appliance. Any design which presents standard security functions via metaphors that are tightly linked to a particular application domain would likely fall into that category of “security appliance” designs. An example would be an electronic mail security program that presented the use of its encryption function as “sending the message via the secured outbox”.

Looking at the problem of creating usable security from a wider perspective, however, it is clear that there are significant benefits to supporting users in developing a certain base level of generalizable security knowledge. A user who knows that, regardless of what application is in use, one kind of tool protects the privacy of transmissions, a second kind protects the integrity of transmissions, and a third kind protects access to local resources, is much more empowered than one who must start fresh in each new application context. Because of this, it is worth considering, as we search for ways to present security functions to users, whether we are better served in the long run by presenting widely used security technologies, such as access control lists and public key cryptography, as tools that can be generally applied rather than as appliance components that appear differently in each application where they are encountered, even if the price for this is to have to support an expanded degree of security visibility in the design task at hand.

## 2.5 The well-in-advance principle

The concept of “just-in-time” help has become a popular usability design strategy [Collins97, Mandel97]. Its prescription is that the coaching necessary to enable a user to perform a particular task should be triggered when the user begins to attempt that task. This is a fine strategy when the task is the user’s primary goal, or when the task is unimportant, but it is a bad strategy when the task is a secondary goal that must be

attended to in order to accomplish the primary goal safely, as is very often the case in computer security.

To see why this is, consider the well-known and frequently lamented tendency of users, when presented with a dialog requesting them to grant some risky permission, to blindly press the “Okay” button and proceed. It should not surprise us that, when users are already focused on some primary task, they will be reluctant to grant much attention to an interruption that tells them they must now learn some new concepts before they can proceed safely.

For security, then, we need a principle of “well-in-advance” help. When some primary user task requires that some security tasks be attended to in order to be safe, then the user needs to have a reasonable idea of the complexity and effort required to achieve those security tasks, well in advance of deciding to tackle that primary task.

## 2.6 Summary

In this chapter, we have presented an analysis of some important ways in which the usability requirements of computer security differ from those of most other types of software applications. We have presented a set of rules and an accompanying design philosophy for determining when it is valid and desirable to try to make security invisible, and have identified a general design principle that is part of the groundwork for the techniques we will present in later chapters.

## CHAPTER 3 Case study: PGP 5.0

### 3.1 Case study goals

In order to get some concrete data to back up our analysis, we looked for an existing software 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. The best candidate appeared to be PGP 5.0 for the Macintosh. Its marketing literature stated that the “significantly improved graphical user interface makes complex mathematical cryptography accessible for novice computer users,” and it was frequently mentioned on computer security discussion lists as an example of security software with a nice graphical user interface.

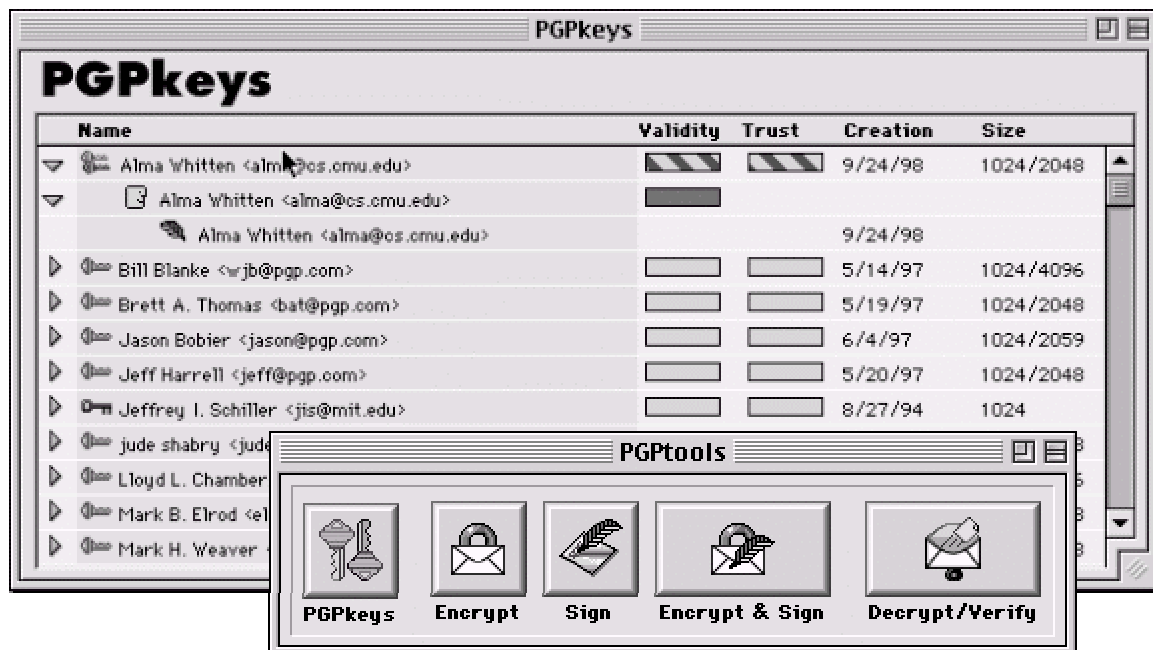


Figure 3-1: PGP 5.0 for the Macintosh

We conducted a formal user test of PGP 5.0, designed to determine whether it was in fact usable enough that people who were reasonably experienced with computers and electronic mail, but unfamiliar with computer security, could successfully use it to send and receive secure electronic mail within a reasonable period of time. A further goal of the study was to determine whether any usability problems evidenced in PGP 5.0 appeared to be standard usability failures or whether they were better explained as failures to design to the special usability requirements of security.

## 3.2 Methodology

We next describe the standard of usability that PGP 5.0 would need to achieve, our user test design for evaluating its success at achieving that standard, and our set of test participants.

### Usability requirements

The first step in our test methodology was to create an informal statement of the usability standard that PGP 5.0 would need to achieve in order to enable people to successfully use it to secure their electronic mail. At minimum, PGP 5.0 would have to be usable enough that its users would be able to do the following:

- 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.

## 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 me 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<sup>3</sup>.
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 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

---

<sup>3</sup> 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 this test was 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.

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.

## Test 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 Appendix A.

## 3.3 Results

For the purposes of this chapter, we summarize the most significant results observed from the test sessions, again focusing on the usability standard for PGP as stated in 3.1. Detailed transcripts of the test sessions can be found in Appendix A.

### 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.

### Basic encryption

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 PGPPKeys. Another of

the twelve (P2) took more than 30 minutes<sup>4</sup> 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.

## Using the correct key

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.

## Basic decryption

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 own 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

---

<sup>4</sup> 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.

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 others’ 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.

## Creating digital signatures

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 digital signatures

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.

## Revocation

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.

## Trust evaluation

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.

## 3.4 Conclusions

The results of the user test appear to support my thesis 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 the area. The 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.

Many test participants did not understand the necessary conceptual model of public key cryptography well enough to be able to figure out which public key to use when encrypting a message for a particular recipient. Many participants who knew which key

they needed to use still had substantial trouble figuring out how to acquire it. A substantial minority of participants accidentally sent their secret messages without encryption, and many participants expressed serious frustration to the point where it seems unlikely that they would have persisted in trying to use PGP 5.0 if they had been doing so in their own lives rather than in the test session.

## CHAPTER 4 Design technique: safe staging

Chapter 2 of this dissertation discussed some reasons why invisibility may not be a valid or desirable design strategy for creating usable security, and Chapter 3 demonstrated some of the ways in which an apparently user friendly interface can fail to make a computer security technology usable. In this chapter, we will present the first of two design techniques that we will argue are generally applicable to the challenge of making computer security usable while retaining visibility. This technique is an elaboration of the idea of *staged user interfaces*. We will demonstrate that it can be a powerful method for bringing users to an intuitive understanding of the usefulness of computer security mechanisms that would otherwise seem foreign and opaque.

### 4.1 A brief history of staged user interfaces

A staged user interface is one that is intentionally designed to shepherd the user through a sequence of stages of system use, in order to increase user understanding and protect against errors. [Carroll84] introduced the idea of a *training wheels* user interface, modifying a commercial word processor to create an initial stage that disabled those functions that led to common new user error states. Results of user testing that asked novices to begin with the initial stage and then gave them eventual access to the full program showed significant gains in performance and understanding. That paper and its several related publications are almost universally referenced in surveys of human computer interaction literature, but little research has been done that builds on it directly.

More recent examples of research on staged user interfaces, such as [Guzdial95], come from the domain of educational and tutoring software, where the concept of *scaffolding* has been brought in from educational theory. In scaffolding, the user interface or instructor supports the novice user through active help and examples that are gradually withdrawn as the user acquires skill. The withdrawal process is referred to as *fading*. The existing research on software scaffolding has primarily remained within the domain



of explicitly pedagogical software<sup>5</sup>, in which the user is intentionally participating in a tutorial experience, and thus the techniques developed are not designed to generalize to other types of software.

## 4.2 Adapting staging for computer security

Security is often too complicated, but simplifying by hiding security mechanisms from the user carries its own risks and drawbacks, as we discussed in Chapter 2. Fortunately, staging, if properly designed, can allow us to reduce the initial complexity that the user must face, without sacrificing either immediate safety or the visibility that will be needed for informed use. There are many ways to interpret the concept of staging, so we will begin by defining a number of terms that can be used to classify different staging methods, and will briefly discuss some of the strengths and weaknesses of each of the resulting classes.

In any staged user interface design, there is the question of how and when the user's progress to the next stage is triggered or permitted. The first pair of terms that we define classifies staging on the basis of whether it is the software that enforces when and whether the user may progress, or the user who decides when progress is desired:

**hard staging:** a user interface design in which the software explicitly enforces a set of requirements for when and whether the user may progress to the next stage, possibly via the implementation of training wheels or a required interactive tutorial.

**soft staging:** a user interface design that allows the user freedom to decide when to progress to the next stage, and encourages progression by establishing a context in which it is a conceptually attractive path of least resistance.

It is generally not realistic or appropriate to design hard staging into security for consumer software. That kind of explicit control over the user's options and progress is acceptable for business software, where an employee may be directed to work through a training wheels phase, and it is expected in educational software that functions in the role of a tutor or classroom instructor. It is unlikely to be appreciated by consumers who are using software on their own computers for their own purposes. We will thus be focusing here primarily on the design of soft security staging.

The next useful distinction classifies security staging on the basis of what is restricted in the beginning stage or stages, using the following pair of terms:

---

<sup>5</sup> Scaffolding has also become a sufficiently fashionable term that it is often used to refer to any form of interactive help built into a software program, but we have not been able to locate any serious research on how to implement real software scaffolding outside of an explicitly educational context.

**function restricted staging:** a user interface design in which the use of potentially dangerous functions is avoided until a stage when the user is competent to manage the security necessary for protection.

**data restricted staging:** a user interface design in which exposure of private data or other valuable resources to the software is avoided until a stage when the user is competent to manage the security necessary for protection.

The biggest obstacle to function restricted staging is the fact that, as security designers, we are often attempting to enable the adoption of security by people who are already accustomed to using the dangerous functions without protection. Obviously, we cannot realistically expect users to accept security software that asks them to suspend or even delay their access to instant messaging, downloadable executables and electronic mail, yet those are some of the applications for which we would most like to design truly usable security.

Another reason why we may often prefer data restricted staging over function restricted staging is that the former permits the possibility of encouraging people to use dummy data in order to explore the potentially dangerous functions and the security needed for protection. Function restricted staging does not offer a comparable opportunity for safe exploration and practice. It may still be useful for some applications, however, either alone or in combination with data restriction.

## 4.3 The design technique

In this section we will discuss how to determine whether a particular security mechanism within an application is a good candidate for staging, and, once a candidate security mechanism has been identified, how to design soft staging which is effective and safe. We will also briefly discuss intermediate stages and multiple staging paths and the types of applications for which each is most likely to be useful.

### Deciding which security mechanisms to stage

Given a particular security mechanism, the following guidelines are useful for deciding whether it is an appropriate candidate for staging:

- Can it be presented in a clear and usable way without staging? If so, then there is little benefit to staging it.
- Is it expected that, once the user is comfortable with the mechanism, they will be making active use of it on a constant basis? If so, then it probably should be visibly represented in the main program window, which means that staging will not offer as much reduction of initial complexity as it otherwise might.

- Can an initial stage be designed that safely postpones the user's need to understand the security mechanism, in accordance with the requirements described in the next section? If not, then it cannot be safely staged.

## Creating soft stages that are safe

A security mechanism can be safely postponed by a stage if the user interface can be designed to meet all of the following requirements as soon as the stage begins:

1. The user knows which of the available actions are dangerous.
2. The user knows what bad consequences might result from the danger.
3. The user understands how to use a temporary strategy for avoiding the danger.
4. The user understands how to learn about the postponed security mechanism when the limitations imposed by the temporary strategy become unacceptable.

Meeting these requirements does not present a difficult design problem; in fact, this set of requirements corresponds closely to the ANSI specification for a standard consumer product warning label [ANSI98], so it is reasonable to expect that the necessary information can be successfully conveyed in a brief and immediate fashion. Furthermore, presentation of this amount of information is unlikely to significantly delay or annoy a user who already understands the security mechanism.

The temporary strategy for avoiding the danger may be based on either data restriction or function restriction, keeping in mind that data restriction also has the desirable result of supporting learnability via the ability to practice safely with dummy data. The success of the security user interface as a whole also requires that the security mechanism, once the user decides to investigate it, be presented in a usable and learnable manner, but that is true regardless of whether or not staging is used. Finally, note that the requirement of these points at the beginning of the stage is in accordance with the well-in-advance principle discussed in Chapter 2, so that the user is never surprised by a warning that the action they are about to take is dangerous.

## Intermediate stages and multiple staging paths

Intermediate stages are most likely to be appropriate in the case when multiple security mechanisms combine to provide protection for one dangerous action. In such a case the staging might be designed to introduce an additional security mechanism at each stage, reestablishing the safety requirements to reflect the danger that remains. An example of such a design will be discussed in the next section.

An application that contains multiple security mechanisms for protection against multiple dangers might conceivably have multiple, separate staging paths. Few applications present that degree of security complexity, but general operating system level security certainly does. A detailed investigation of how to create such staging is beyond the scope of this thesis, but may be addressed in future work.

## 4.4 Example: staging applet security

An interesting example of what may be considered to be a sort of staged security user interface can be seen in the HotJava web browser [Sun95, Fritz98]. When Java applets were first introduced, it was expected that they would be used for a wide variety of non-trivial applications, to the point that there were predictions that web browsers would, in effect, replace desktop operating systems. Some applets would be harmless bits of web page decor, but others, accessed by subscription at the software publisher's web site, would serve as word processors, desktop publishing software, and checkbook balancers. This prospect required security that could allow known, authorized applets to access resources on the user's machine such as files, directories and printers, while blocking randomly encountered applets from doing anything dangerous or perhaps even from executing at all.

### The HotJava applet security user interface

In HotJava, the user is not initially presented with any information about applet security. The default security configuration, which may be considered to represent the initial stage, allows all applets to run, but does not give them access to local system resources such as files. Applets that are digitally signed, however, can request access to those resources by invoking a dialog that asks the user to grant permission.

Figure 4-1 shows the Advanced Security Administration Mode of the Security Preferences dialog, with the default settings still in place. A user who is viewing this screen may be considered to be in the second stage, in which they have the option of choosing from several predefined security policies to assign to signed applets and to unsigned applets, still as two indivisible groups.

Finally, in the third stage, the user explores the Special Cases Panel, which allows the creation of detailed, individual security policies to be applied to applets signed with particular keys.

This security user interface has an immediate serious problem which can be directly related to violation of the requirements for safe staging. In the initial stage, the dangerous action is that of running an applet with access to system resources such as the user's private files. The user is unlikely to see any information about this danger until the first time that a signed applet requests access permission, at which point they would have to immediately learn the basics of digital signature based authentication and make an

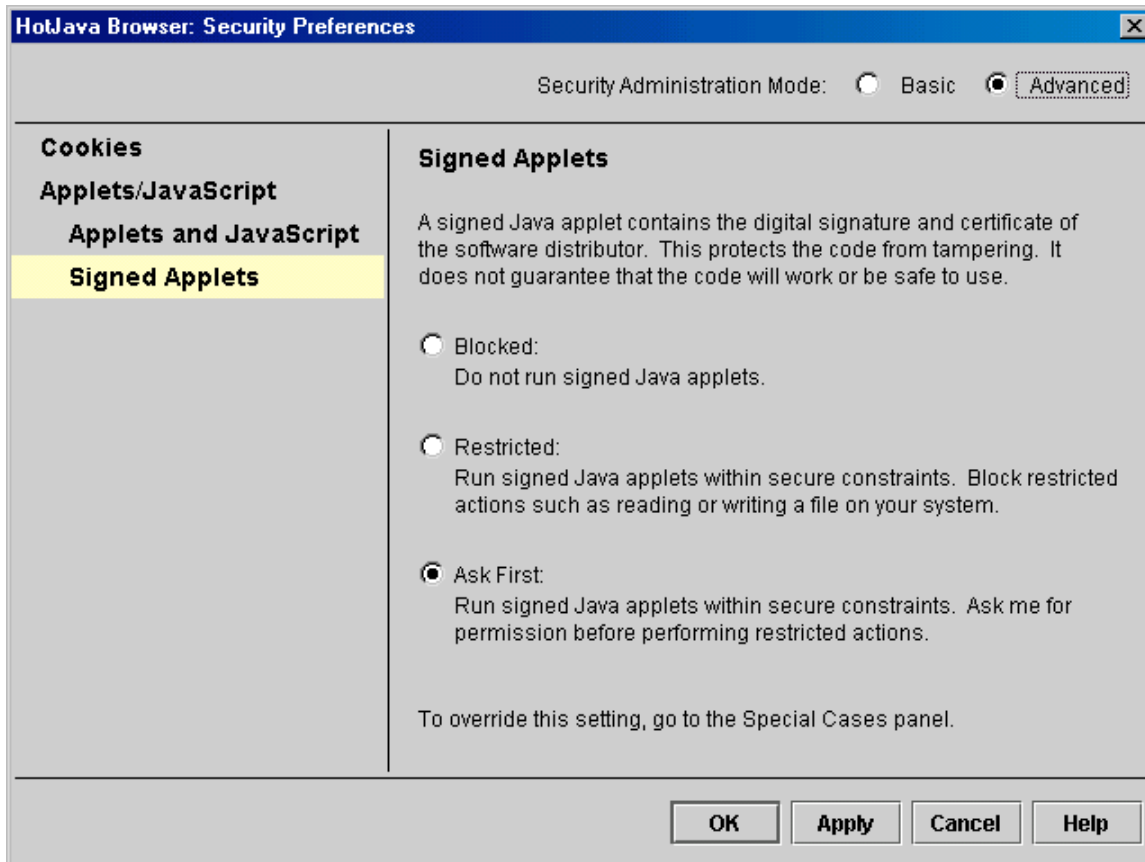


Figure 4-1: Applet security management in HotJava

informed judgement about the risks of granting the access. This violates most of the safe staging requirements as well as the well-in-advance principle, and will almost certainly lead to the blind granting of dangerous permissions by users.

## An improved design using soft, data restricted staging

By contrast, following the design method for safe soft staging yields a much more attractive solution. There are two security mechanisms here: digital signature based authentication, and access control lists. They are combined to provide protection for the dangerous action of running an applet with access to local system resources such as the user's files. This suggests a design with three stages: initial, intermediate, and final.

A data restricted strategy can make it safe for the user to postpone learning about either of those mechanisms, if the default security policy is set to allow applets to access only a single, dedicated directory that has been created for that use alone and contains no preexisting files. The initial stage would then be designed to meet the safety requirements by conveying the following to the user:

1. That allowing an applet access to their data is dangerous.
2. That an applet might steal, damage, or otherwise misuse their data.
3. That they can avoid the danger by withholding any data that they consider valuable or private both from the dedicated applet directory and from any direct interaction with the applets.
4. That when they wish to consider whether a particular applet may be trusted with certain valuable or private data, they can investigate the digital signature based authentication mechanism.

In the intermediate stage, the user would have the option of granting a digitally signed applet its own dedicated directory, similarly created for that use alone, and accessible only by applets signed with that particular key. Then the new set of safety requirements would require the following to be conveyed to the user:

1. That allowing an applet access to data that it did not store itself is dangerous.
2. That an applet might steal, damage, or otherwise misuse data it does not own.
3. That they can avoid the danger by continuing to withhold any data that they do not specifically want a particular applet to have.
4. That when they wish to allow an applet limited, specific access to data outside its own directory, they can investigate the access control list mechanism.

In the final stage the user would be able to define fine grained access control policies for applets signed with particular keys, just as they would in the final stage of the original HotJava security user interface, but in this design they would have travelled a much safer and more plausible path to arrive at that destination.

## 4.5 Variant: staging for the abstraction property

It is also interesting to consider how staging might be used to address the abstraction problem discussed in Chapter 2. That is, we recall that security is often confusing to novices not only because of the number of new mechanisms to be understood, but also, and perhaps even more importantly, because many security mechanisms represent pared down abstractions of messy, human social processes such as trust, identification and permission.

One way to help users become comfortable with those abstractions might be to design an initial stage that encourages the use of a temporary protection strategy that combines data restriction with an intuitive, social approach to protecting against the danger. This would allow users to practice thinking about the process for which the security mechanism is an

abstraction, preparing them to recognize the mechanism itself as sensible and useful once they decide to investigate it. The intuitive, social approach will probably provide only weak security at best, but that is not a safety issue as long as it is combined with data restriction and accurate presentation of the remaining danger, in accordance with the safe staging requirements. In the remaining section of this chapter, we present a detailed design and implementation of this type of staging strategy as applied to the highly abstract mechanism of key certification.

## 4.6 Safe staging demonstrated in Lime

In order to demonstrate and evaluate the success of both the staging design method presented in this chapter and the metaphor design method that we will present in Chapter 5, we designed and implemented a complete software simulation of a user interface for a public key based secure electronic mail application, which we named Lime. In the remainder of this chapter, we will present and discuss the components of Lime that implement soft staging of the mechanism of key certification. This will necessarily involve some exposure to the set of visual metaphors used to present public key cryptography in Lime; those visual metaphors will be fully presented and discussed in Chapter 5, along with the rest of the Lime user interface design. Evaluation of the staging, the metaphors, and the overall success of the Lime design will be covered in Chapter 6.

In Lime, only key certification is staged; the mechanisms of encryption and basic digital signatures are not. It would have been possible to design an interface in which those mechanisms were also staged, with an initial stage in which the user would send only plaintext email, using data restriction as a protection strategy, and investigating the use of cryptography at the point when they decided they wanted to send something private in a mail message. However, the other two guidelines for deciding whether or not to stage a security mechanism suggested that it was better not to stage them: first, it is possible to present them in a clear and usable way without staging, as will be shown in Chapters 5 and 6, and second, they may be actively used whenever an email message is sent and thus should be visible in the main window, which limits the advantages of staging them. Key certification, by contrast, is extremely confusing and difficult to present clearly, and does not need to be visible in the main window since we expect that it will be used only when public keys are traded.

Key certification is confusing in large part because of the abstraction problem, so my staging design incorporates a strategy designed to familiarize the user with the process of trying to verify the identity of a correspondent who can't be met with in person. In the absence of a security mechanism for verifying identity, an informal, intuitive strategy is to look for familiar writing style and mention of personal information that an imposter would be unlikely to know. For the purpose of this discussion, I will refer to that strategy as *social authentication*.

In staging key certification, the initial stage may be considered to begin when the user decides to use cryptography and needs to attend to the prerequisite task of trading public keys. At that point, the user interface must meet the following safety requirements:

1. The user knows that sending private data in email is dangerous if you are not sure that you have the right public key to encrypt it with.
2. The user knows that an attacker might be able to spy on them if they use the wrong public key.
3. The user knows that they can get some assurance that they have the right public key if they use social authentication when trading keys by email, but that it is only enough to protect against a random attack.
4. The user knows how to investigate key certification when they decide that they need stronger security.

As previously mentioned, Lime begins with an initialization sequence, the first screen of which was depicted in Chapter 1, Figure 1-1. The second screen helps the user generate a key pair, and the third screen helps them store it securely with a pass phrase. The fourth and fifth screens introduce digital signatures and encryption, and will be shown in Chapter 5. The sixth screen gives a quick briefing on the need to trade public keys, as shown in Figure 4-2. In its first paragraph it presents the information necessary for the first and second safety requirements.

Figure 4-3 depicts the Lime key trading screen, which reiterates the danger, establishes and reinforces a context in which methods for trading public keys are associated with the level of security that they provide, and mentions key certification for the first time, using the metaphor of *certification locks*. It steers the user toward safe practice with social authentication by setting up trading public keys by email as the easiest and most inviting option, with a helpful form to use in order to perform it safely. While it is possible to trade keys directly in email without first having recourse to this screen, the Lime interface has been designed to shepherd the user toward this screen at every possible opportunity. It is accessible directly from the last screen of the initialization sequence, using the button marked “Trade keys now”, from the Help menu on both the main window and on any message composition window, under “Help on trading public keys”, and from the Public Key Rolodex using the large button that resembles a space bar and is labeled “Need a key that isn’t in your rolodex?” as shown in Figure 4-4.



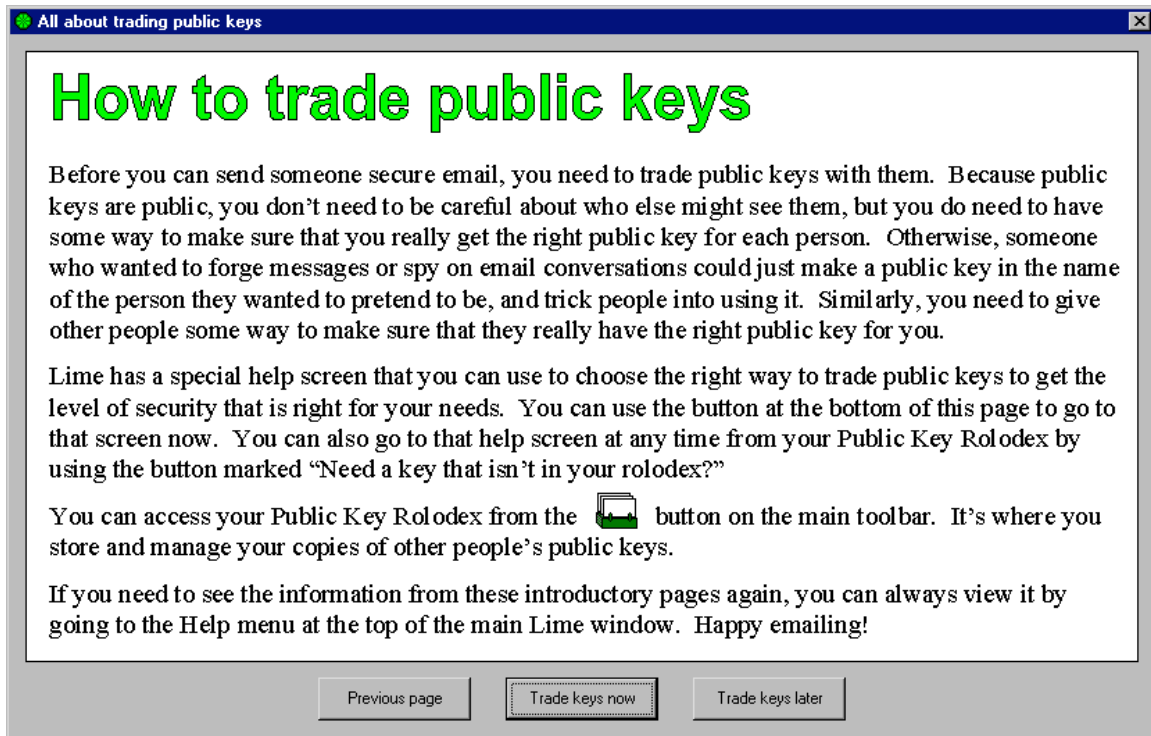
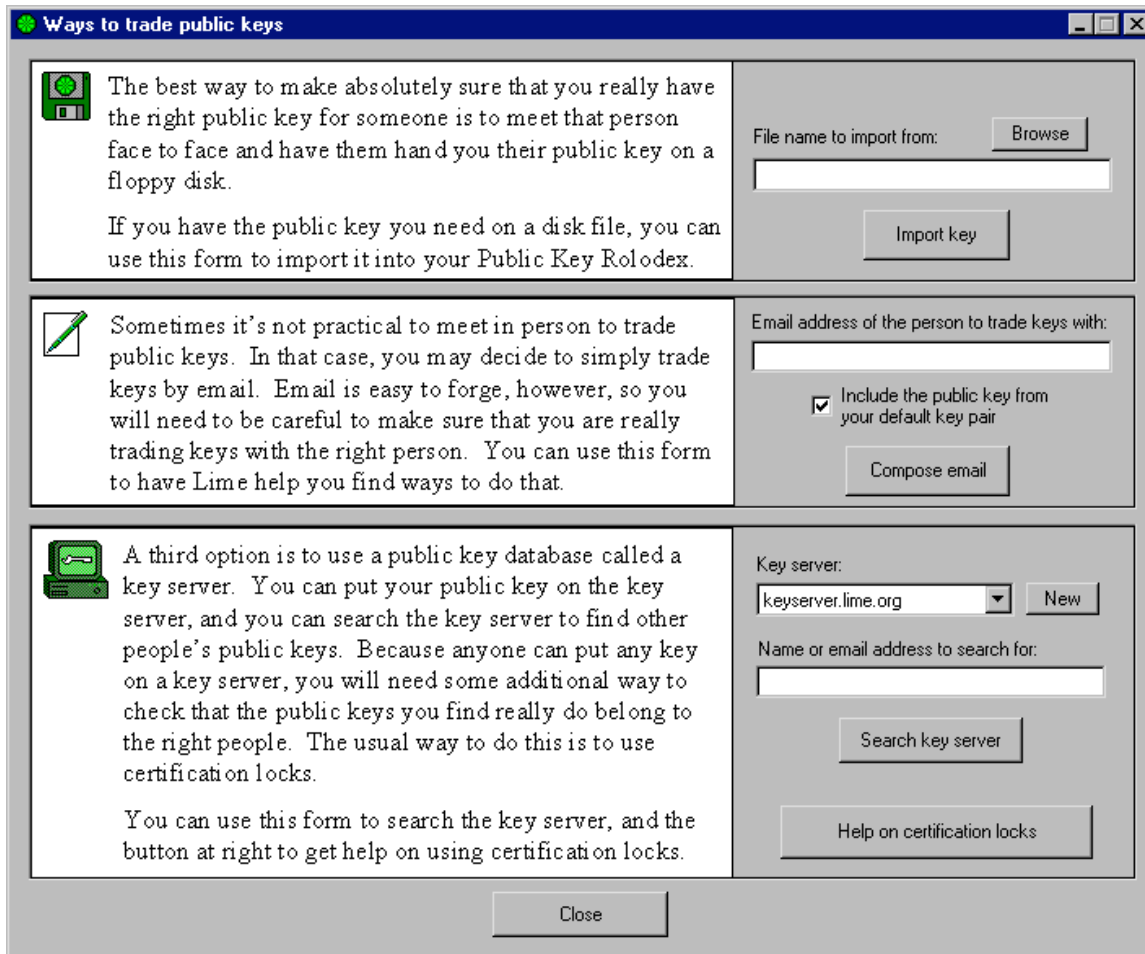


Figure 4-2: Sixth and last screen in the Lime initialization sequence

Use of the "Compose Email" form causes the help screen shown in Figure 4-5 to display, accompanied by a fresh message composition window like the one shown in Figure 4-6. This help screen, which is also always available from the Help menus under "Help on trading public keys by email", details the use of social authentication, and again offers the opportunity to investigate certification locks if stronger security is needed. At this point all of the information specified in the safety requirements has been conveyed.

This set of screens contains rather a lot of text, but the information that is necessary to the safety requirements is generally contained in the first few sentences, and the gist of the dangers and the protection strategy can be gathered from even a cursory scan.

When the user is ready to investigate certification locks, they can access the screen shown in Figure 4-7 from the public key trading screen, from the Help menus, or from the help screen for trading public keys by email. Further information on how to get one's own key validly certified and how to validly provide certification to others is given in the additional screens shown in Figure 4-8 and Figure 4-9, respectively.



**Figure 4-3: The public key trading screen**

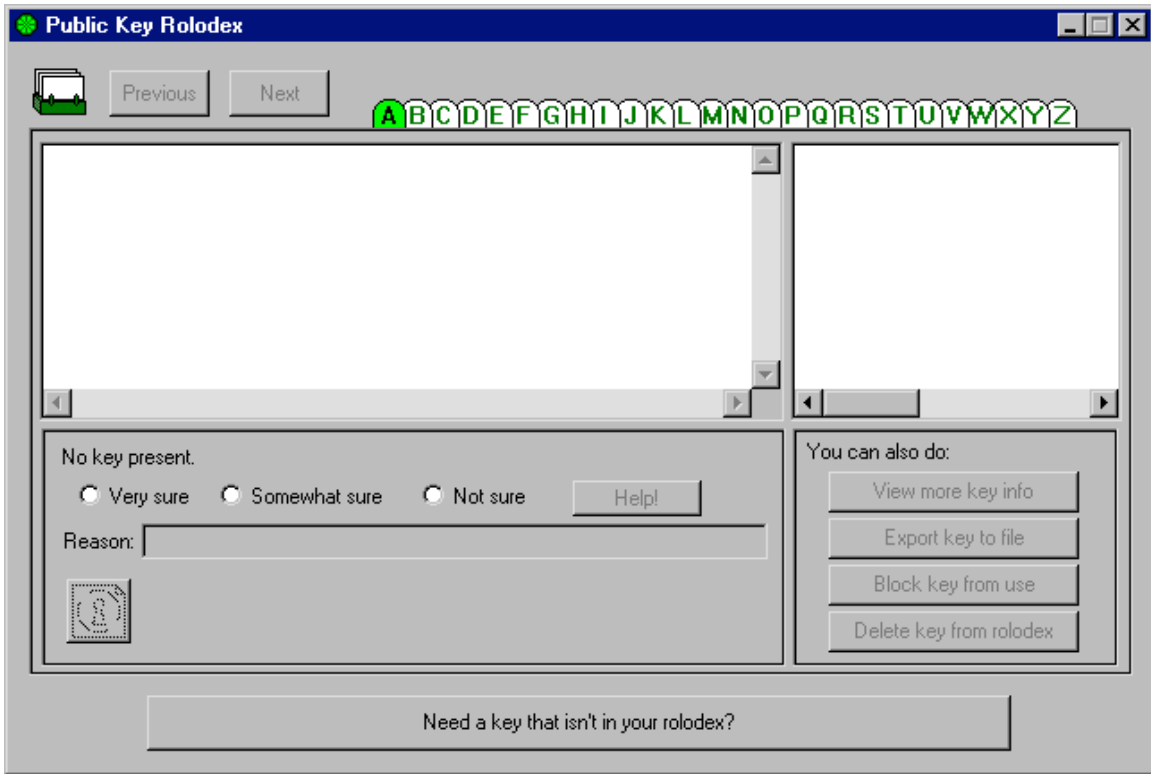


Figure 4-4: The public key rolodex

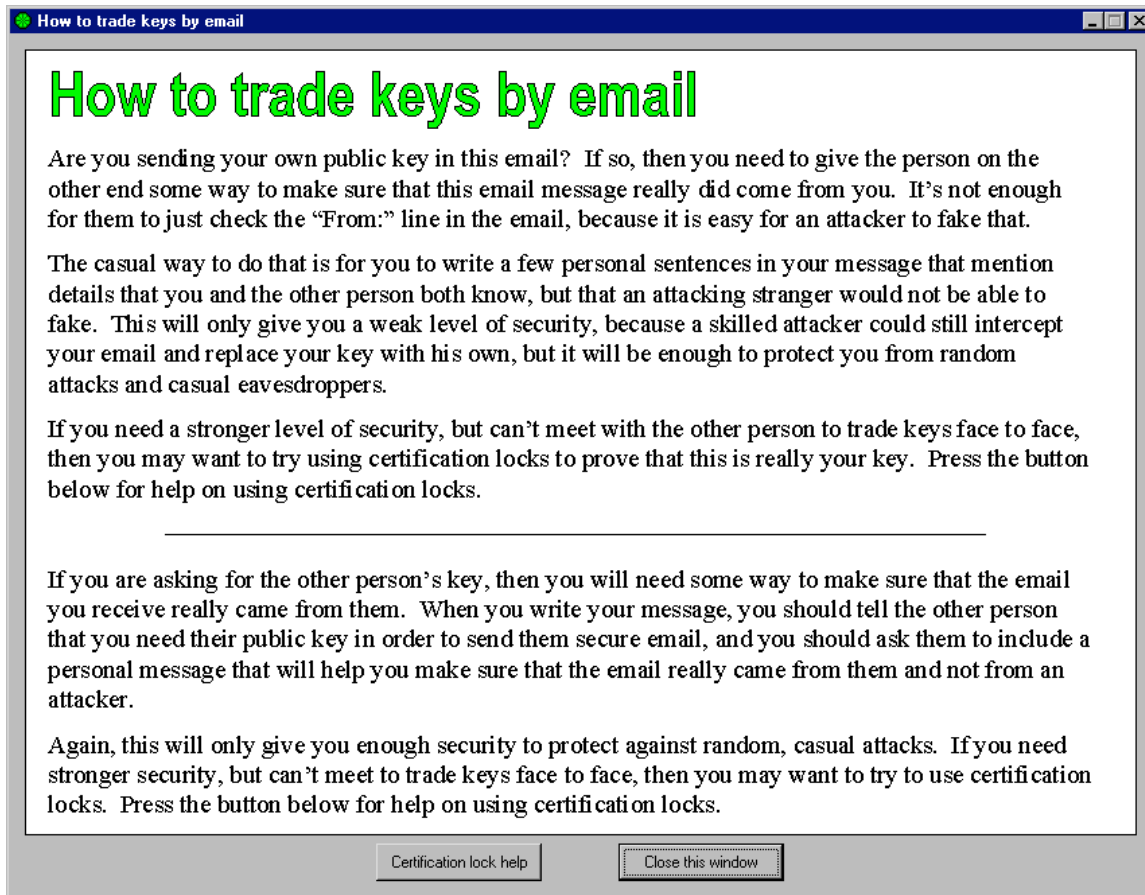
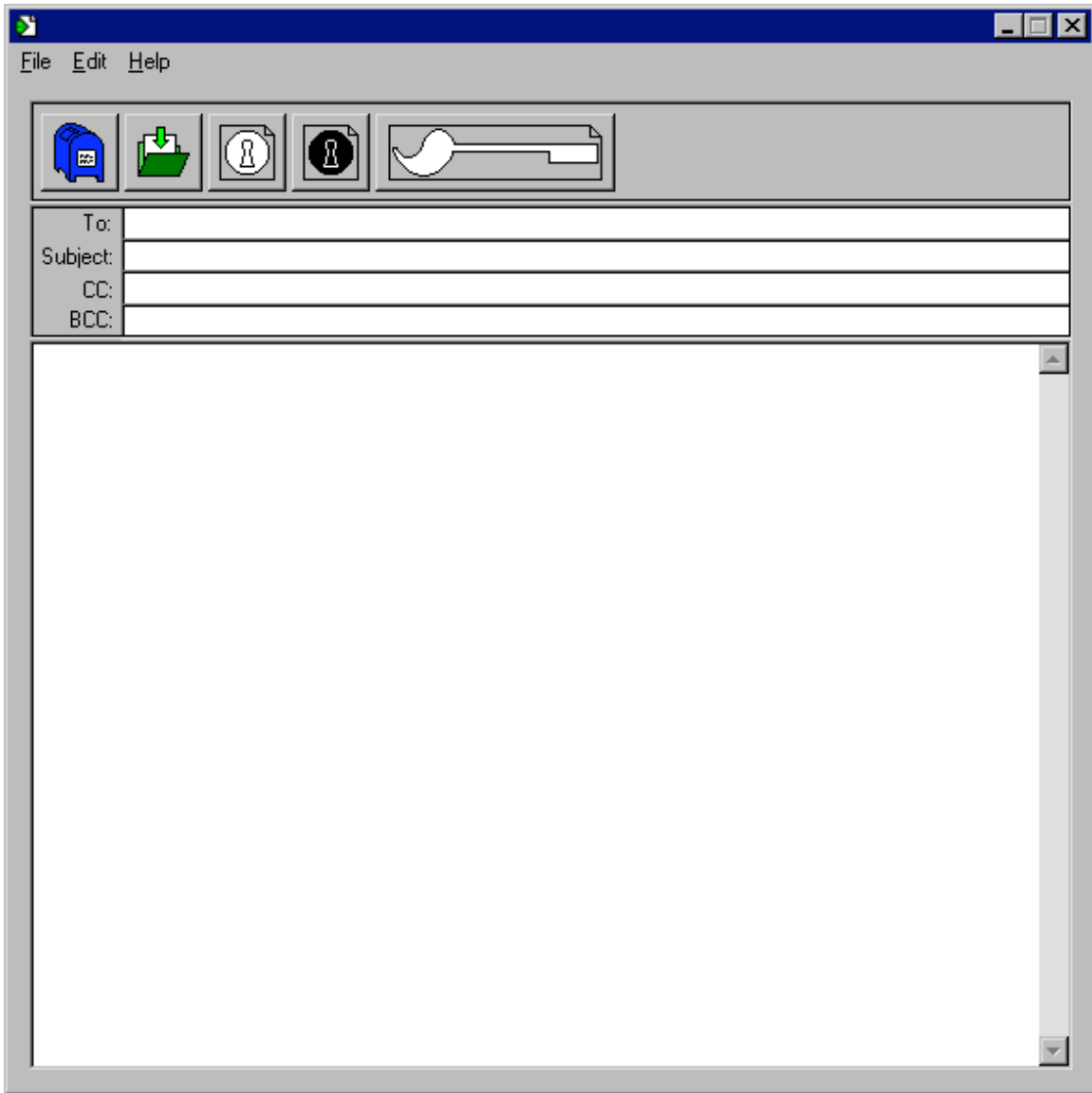


Figure 4-5: Safe presentation of social authentication strategy



**Figure 4-6: A blank message composition window**


All about certification locks

# All about certification locks

A certification lock is a special kind of content lock that is made to be put on a public key. It has a very specific meaning: it states that the person who created the lock has made absolutely sure that this public key really does belong to the person whose name is on it.

This is what a public key with a certification lock on it looks like:

On this public key certificate, the certification lock means that Trusted Certifier claims to have personally verified that this public key really does belong to Kelly Green.



If you get someone's public key with a certification lock on it, then you will need to consider two questions. First, do you trust the person who applied the certification lock to have really verified the owner of the public key? Second, are you absolutely sure that you have the right public key for the person who applied the certification lock? If the answer to both those questions is yes, then you can be confident that the certification locked public key really does belong to the person whose name is on it, and you can use it to get a high level of security for your email messages.

As you can see, in order to use certification locks, you need to already have at least one person's public key that you can be absolutely sure of, so that that person can be the trusted certifier for other people's public keys.

Use the Help menu or the buttons below to get instructions on getting and creating certification locks.

[Help on getting certification locks](#)
[Help on creating certification locks](#)
[Close](#)

Figure 4-7: Presentation of certification lock mechanism

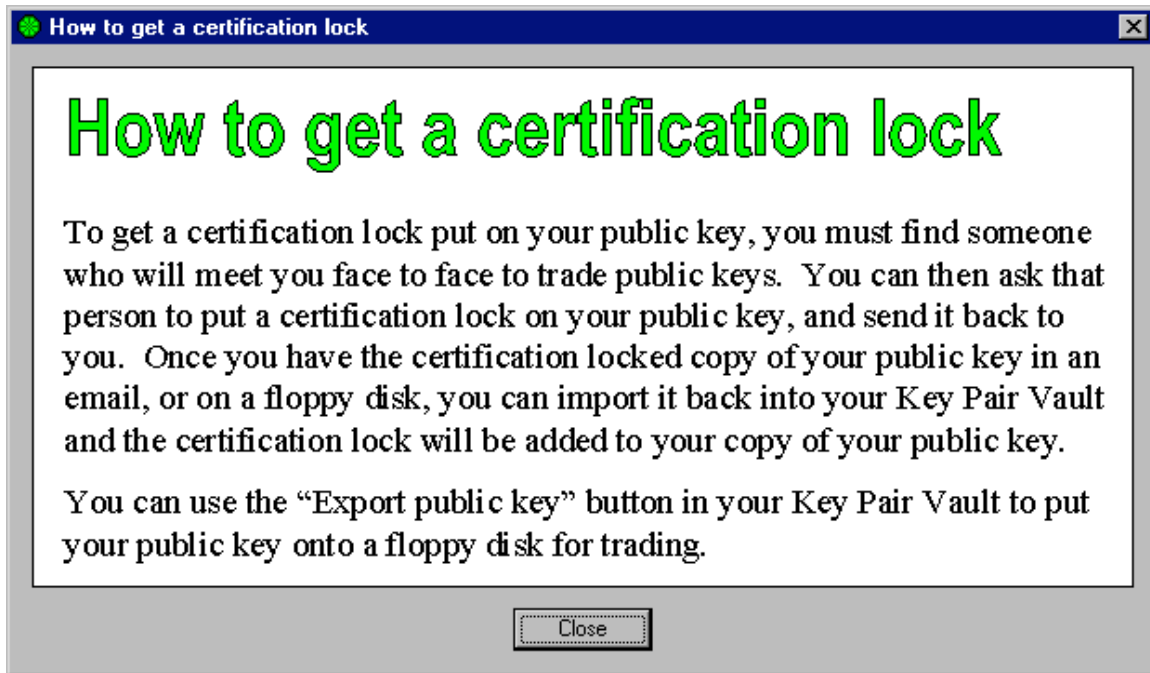


Figure 4-8: Guidance on getting a certification lock for one's own key

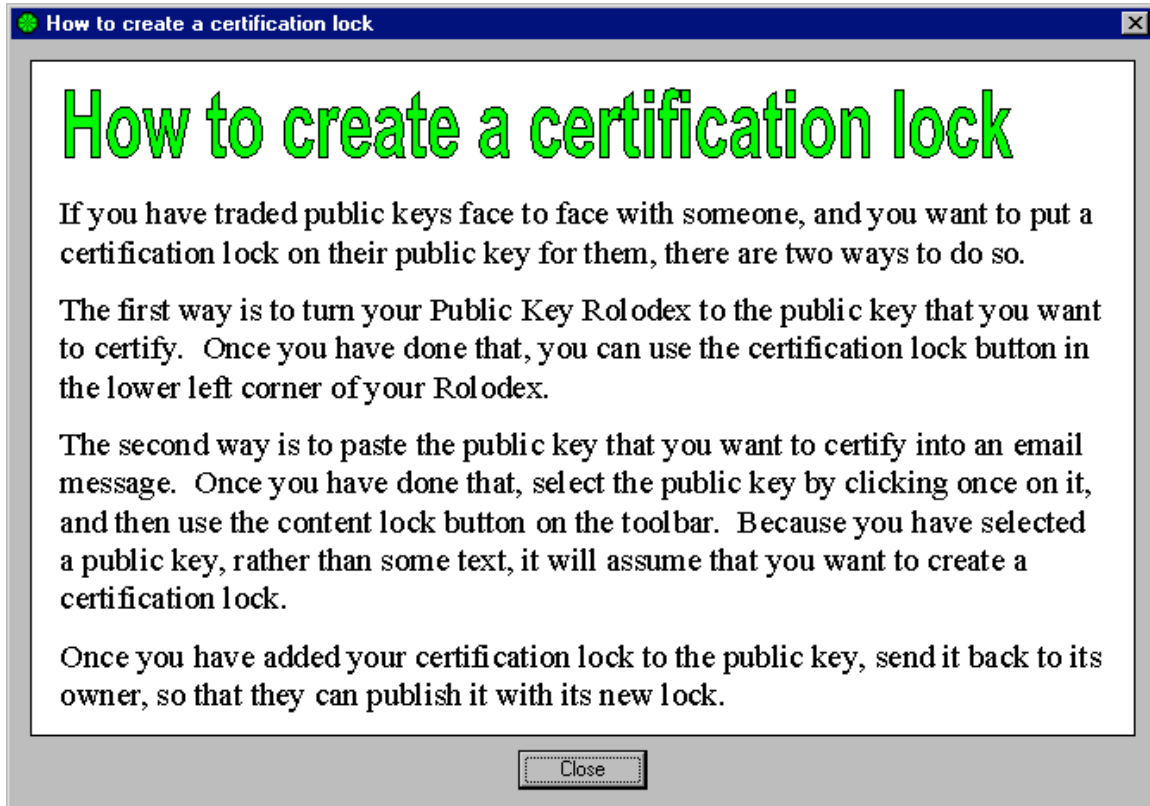


Figure 4-9: Guidance on creating a certification lock for someone else's key

We have now presented and described the implementation of safe staging for key certification in Lime. In the next chapter, more images of Lime will be presented and discussed as we describe and demonstrate our second design technique. Chapter 6 will then present experimental results for the effectiveness of the design techniques and the usability of the security in Lime.



## CHAPTER 5 Design technique: metaphor tailoring

In this chapter we present our enhanced and specialized design technique for creating graphical representations of security state and mechanisms. Extensive literature already exists on how best to design user interface metaphors and their graphical representations, but really good visual metaphors for security have remained elusive. We suggest that this is because such metaphors would need to map very tightly to the information that is most crucial for the user to be aware of in order to maintain their security, and the techniques outlined in the existing literature do not provide designers with enough guidance to help them target the right mapping. This leads to faulty metaphors such as the digital signature with quill pen that we saw in PGP 5.0, which did nothing to evoke the actual security provided by digital signatures.

To remedy this, we take a well-known design technique, *conceptual model specification* [Liddle96, Johnson02], and enhance it to explicitly capture the security information that is most important for the user to understand. This enhancement is somewhat similar to the standard security practice of specifying an explicit threat model, but is focused on the risks of usability failures rather than on the risks of external attacks. We call this enhancement *risk enumeration*. We then describe how to design visual metaphors that map tightly to these enhanced conceptual model specifications, using standard user interface design heuristics but prioritizing them toward qualities like reassurance and accuracy which are particularly important for usable security. We refer to this entire enhanced and prioritized process as *metaphor tailoring*. We conclude this chapter with an example of firewall port management as a second illustration of metaphor tailoring.

### 5.1 Risk enumeration

Many usability engineers recommend the creation of an explicit conceptual model specification before any user interface screens or dialogs are designed. A conceptual model specification is generally considered to be a high-level description of the data and

functions that are presented to the user, and of the conceptual objects or mechanisms that are used to represent them. This specification can then be used, when designing the actual user interface, as a guide to make sure that those concepts are presented clearly, consistently and coherently.

In the case of security, one of our most imperative user interface design goals is to avoid usability failures that cause the user's private data or other resources to be exposed or damaged. We can extend our conceptual model specification to explicitly support that goal by including an enumeration of the possible risks of user misunderstanding that our design must guard against. To generate such an enumeration in a systematic way, we propose that it is useful to consider these risks as falling into three categories:

**Accident risks** are risks that the user will forget that a security task needs to be attended to, will fail to realize that taking a particular action is dangerous, or will fail to notice that their system is in an insecure state.

**Frustration risks** are risks that the user will find it too difficult or time consuming to figure out how to accomplish a primary task in a secure manner, and will choose to forego security as a result.

**Bad citizen risks** are risks that the user will misunderstand the system in ways that may not necessarily imperil their own security, but which may lead them to damage the integrity of a larger system that they are participants in.

Table 5-1 shows a conceptual model specification, enhanced with risk enumeration, for the secure electronic mail program that we are using as a design example. To develop this specification, we had to first identify the visibility requirements and decide whether to design for a tool or an appliance, as discussed in Chapter 2. We briefly cover each of those points here.

### **Visibility requirements**

Lime is intended to be a secure electronic mail application for general home use, in which messages may be sent to anyone who has a valid email address, regardless of whether they have made a public key available. We consider it unrealistic to assume the availability of a reliable, centralized key certification authority for individual users' public keys, so Lime is also intended to support a "web of trust" model in which users must decide which other users they trust as certifiers of public keys.

Under those circumstances, Lime has the following visibility requirements:

- The presence or absence of encryption on a message must be visible, because it cannot be guaranteed that every message can and will be encrypted, since some message recipients may not be using secure mail software.
- The process of public key acquisition and distribution must be visible, because there are likely to be cases in which getting the public keys necessary to send an

encrypted message is only possible if the user is capable of directly negotiating a key exchange with the intended recipients.

|  |  |
|--|--|
| <p><u>Objects</u></p> <p>Key pairs, each consisting of a secret key and a public key</p>   | <p><u>frustration risks</u></p> <p>Difficulty generating own key pair<br/>         Difficulty protecting own key pair with pass phrase<br/>         Difficulty encrypting messages<br/>         Difficulty choosing correct public key with which to encrypt message<br/>         Difficulty getting other people's public keys<br/>         Difficulty giving own public key to other people<br/>         Difficulty signing messages<br/>         Difficulty signing public keys<br/>         Difficulty getting own public key signed by an appropriate person<br/>         Difficulty verifying signatures on messages or public keys<br/>         Difficulty revoking public keys</p> |
| <p><u>states</u></p> <p>Messages may be encrypted or plaintext.<br/>         Message text may be signed or unsigned.<br/>         Public keys may be signed or unsigned.<br/>         Signatures may be valid, invalid, or unverifiable (when matching public key is not available).<br/>         Public keys may be valid, revoked, or expired.</p>   |  |
| <p><u>operations</u></p> <p>Generation of key pairs<br/>         Protection of key pairs with pass phrase<br/>         Encryption of plaintext messages<br/>         Decryption of encrypted messages<br/>         Signing of text<br/>         Signing of public keys<br/>         Deleting signatures from text or public keys<br/>         Verifying of signatures<br/>         Exporting public keys to files<br/>         Sending public keys to key servers<br/>         Attaching public keys to messages<br/>         Importing public keys from files<br/>         Retrieving public keys from key servers<br/>         Importing public keys from messages<br/>         Deleting public or secret keys<br/>         Revoking public keys</p> | <p><u>bad citizen risks</u></p> <p>Ill-considered signing of other people's public keys</p>  |
| <p><u>accident risks</u></p> <p>Accidental exposure of secret key<br/>         Accidental exposure of private message by emailing it without encryption<br/>         Accidental acceptance of unsigned message as authentic<br/>         Accidental acceptance of unauthenticated public key as authentic</p>  | <p><u>logical model concepts</u></p> <p>Everyone has their own key pair.<br/>         Each person keeps the secret half of their key pair secret.<br/>         Each person gives the public half of their key pair to everyone else.<br/>         Each person verifies the identity of the owner of each public key they are given.<br/>         Any security operation performed using one half of a key pair can only be reversed by using the other half.<br/>         Any security operation that can be reversed using one half of a key pair must have been originally performed using the other half.</p>   |

**Table 5-1: An enhanced conceptual model specification for the Lime secure electronic mail program, with risk enumeration**

- The process of digitally signing a message must be visible if we wish to support digital signatures that count as intentional legal signatures.

### **Generality decision**

Public key cryptography is a very powerful general purpose tool within computer security, and is employed in a wide variety of applications. If key pairs and their basic uses could be established as familiar concepts to most computer users, at the same application-neutral level as files, windows and trash cans have been, that would make it much easier to design effective user interfaces for many different kinds of security applications. We therefore chose to design tailored metaphors for Lime that present public key cryptography as a general purpose tool rather than a dedicated component of an electronic mail appliance.

## **5.2 Metaphor tailoring**

We next want to design visual metaphors that map tightly to the information in our enhanced conceptual model specification and convey it to the user as effectively as possible. To do so, we will rely on well-established design principles and heuristics, but we will pick and choose among them to identify and highlight the ones that are most important for usable security. These will tend to be the ones that are most relevant to observability, predictability and learnability [Nielsen94]; since other generally desirable qualities like efficiency and flexibility are of secondary importance for our purposes. We propose that the following five principles and heuristics should stand as our most important design goals.

### **Provide visual cues that evoke correct assumptions**

[Norman94] presented an enormously influential argument that consumer objects should be designed so that people's first guesses about how to use them would be the correct ones. One of his most compelling examples was that of different designs of door handles that make it visually obvious whether the door should be pushed or pulled to open it, by making it physically easy to do the right thing and physically difficult to do the wrong thing. Designs that have this quality are especially robust, because people's ability to do the right thing does not depend on having a correct logical model of how the object works, but only on whether their instinctive reaction to the object cues them correctly. This is very desirable for security, since it is probably unrealistic to expect that all users will construct a logical model. We want to design our visual metaphors so that important security tasks "look like" the right thing to do at the right time.

### **Support learning correct logical models**

There is a great deal of literature discussing what it means for a user interface metaphor to have a good mapping to the functionality that it represents. We here

cite [Carroll85, Kuhn91, Nardi93, Apple96, Mandel97 and Neale97] as influential and representative examples. Since we think that some users will construct a logical model, we think that it is important that the visual metaphors we develop are consistent with a correct logical model and not a misleading one.

### **Provide a shorthand for communicating about the system**

We want our user interface metaphors to provide the user with a set of cleanly encapsulated and informatively named conceptual objects that can then serve as part of the vocabulary for explaining the more complicated details of the security system. If we can do this well, it becomes much easier to communicate effectively in our help screens and warning messages.

### **Provide visual reassurance**

Most computer users have “security anxiety.” They are predisposed to see computer security as a confusing, complicated matter in which they are doomed to flounder and make fatal mistakes. To ameliorate this, it is important that security mechanisms be presented using images that suggest that they are simple, easy and pleasant to use. One strategy for accomplishing this is to use the clear, bright colors and simple, blocky shapes associated with children’s toys, which also provides the benefit of implying that the user’s “beginner” status is welcomed and expected.

### **Do not mislead**

Security user interfaces must, of course, present the user’s security configuration and settings accurately, but some less obvious kinds of accuracy should also be taken into account. In particular, two pitfalls to be avoided are that of making the security appear more sophisticated or absolute than it actually is, and that of making security mechanisms appear to more exactly analogous to real world objects than they actually are.

To accomplish our metaphor tailoring for a particular security application, we first develop an enhanced conceptual model specification with risk enumeration, and then attempt to design a set of named graphical symbols for those conceptual objects, such that the graphical symbols visually convey or suggest as many of the states, operations, logical concepts and risks in our specification as possible. To do this, we iterate through the following steps<sup>6</sup>:

Step one: Identify the set of conceptual objects to be represented. In doing so, look for potentially useful concept hierarchies, so that variations on a concept may be reflected by variants of a basic graphical symbol.

---

<sup>6</sup> We have not explicitly addressed the importance of preliminary user testing and other forms of evaluation as part of this process, but our intent is that such evaluation should be considered to be an inherent part of steps two and three.

Step two: Construct a basic symbol to evoke the necessary qualities and associations for each conceptual object.

Step three: Design variations and decorations for those symbols to signal context-specific operations, states, and risks.

When it appears that further iteration will not achieve any additional coverage of the conceptual model specification, the set of tailored metaphors is complete. It may be useful to then annotate the conceptual model specification to show which items are well supported by the tailored metaphors and which are not, for later use in designing help screens and other components of the user interface.

## 5.3 Metaphor tailoring demonstrated in Lime

Earlier in this chapter we created the extended conceptual model specification for our example security application, the Lime secure electronic mail program (Table 5-1). We now work from that specification to create our set of tailored visual metaphors.

### Conceptual object set

The first step is to choose the set of conceptual objects to be represented, and to look for potentially useful concept hierarchies. Within the electronic mail application, the important conceptual objects are the key pairs, consisting of a public key and a secret key, the operations of encrypting and signing, and the corresponding states of being encrypted or signed. Encryption and signing can also be seen as part of a useful conceptual hierarchy of cryptographic operations, like this:

```
base concept:  cryptographic operation
  subtype1:   encryption
  subtype2:   signing
    subtype1:  tamperproofing
    subtype2:  signing legally
    subtype3:  certifying a key
```

Developing a base graphical symbol for cryptographic operations that can be varied to depict the different subtypes will be very helpful to the presentation of public key cryptography as a tool rather than an appliance, since it might be extended in future to cover other semantic meanings for digital signatures, or to depict encryption as one of a variety of ways of controlling access to data.

In the next two subsections we will describe how steps two and three of our technique were applied to produce the tailored metaphors first for key pairs, and then for cryptographic operations, the latter of which is then specialized for encryption and for digital signatures, following the conceptual hierarchy outlined above.

## Tailored metaphor for key pairs

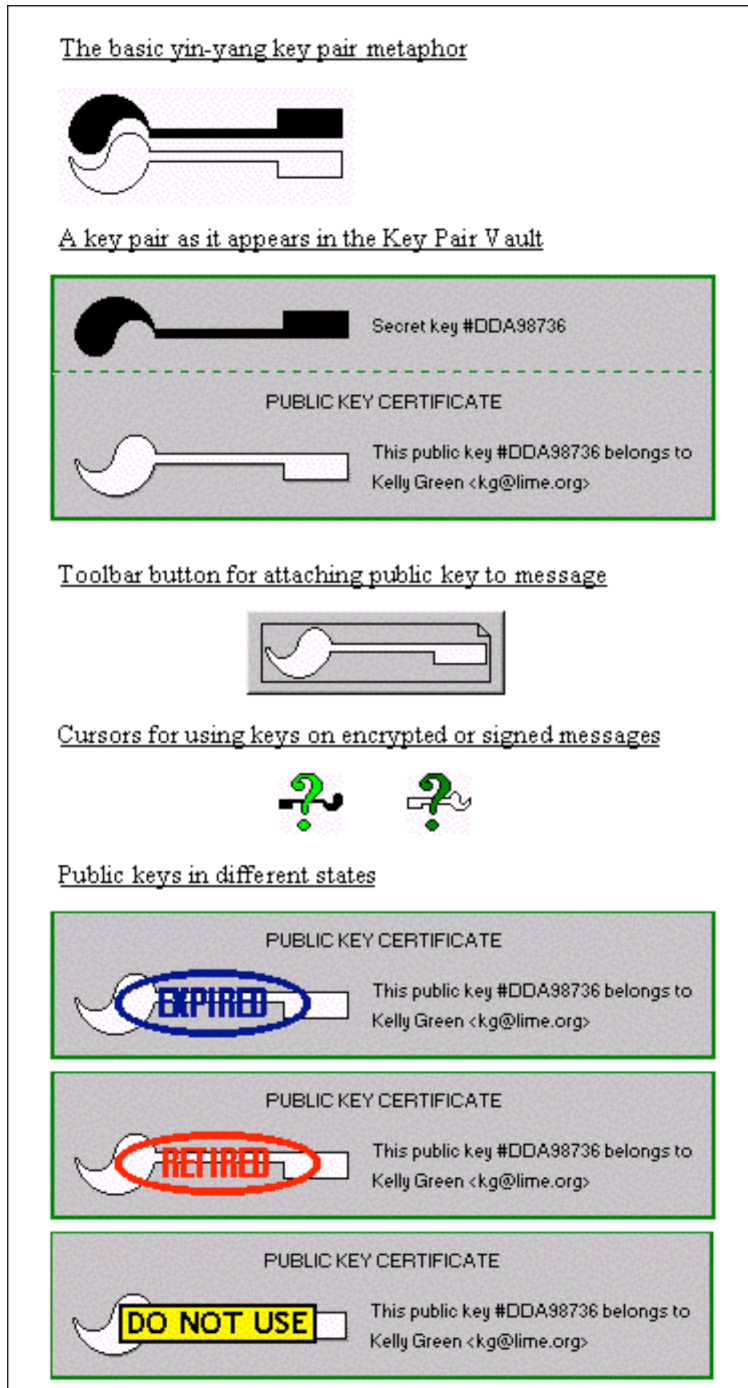
In step two of the metaphor tailoring technique, a basic graphical symbol is constructed to evoke the important qualities and associations for the conceptual object. For key pairs, those are:

- The object is a tool for achieving security.
- The object has two halves that uniquely match each other.
- The two halves of the object have opposite qualities and can reverse each other's operations.
- One half of the object must be kept secret.
- The other half of the object should be given out freely.

The yin-yang key pair image depicted at the top of Figure 5-1 covers all of those points. Both the combined key pair and the individual halves resemble keys, suggesting security. The two halves fit together like puzzle pieces, suggesting the unique match. The two halves are oppositely colored, black and white, and incorporate the yin-yang symbol, which not only evokes oppositeness, but also the appropriate qualities for each half: privacy and hiddenness for the black, secret key, and aggressive activeness for the white, public key. The design principle of accuracy is reflected in the fact that the image, while having a key shape, is clearly a more abstract object than a house key, and the principle of reassurance is met by the use of simple, graceful shapes and primary colors. We retain the names “key pair”, “public key” and “secret key” as reasonably simple and informative.

In step three, the basic symbol is elaborated to signal the important context-specific operations, states, and risks. Another important quality of key pairs is the fact that a key pair belongs to a particular person, and that the identity of that person is explicitly attached to the public key. The next image shown in Figure 5-1 is the elaborated visual symbol for a key pair in certificate form, as it would appear in the user's key pair storage (which in Lime is called the Key Pair Vault). This symbol reinforces the association between the two halves, by repeating the key ID for each, and emphasizes the identification of the owner of the public key, by displaying it in text that takes up as much space as the key itself. By depicting the certificate as having a tearable perforation separating the top part, with the secret key, and the bottom part, with the public key, it neatly provides for the separate depiction of public keys when they are attached to messages or stored elsewhere.

This leads nicely to the next image in Figure 5-1, which is the toolbar button used for the operation of attaching a public key to a mail message. The public key embedded in a rectangle of gray is a direct depiction of the public key certificate image, and the folded over corner, evoking the widely used icon for creating a new file, helps to emphasize that a “new” copy of the public key certificate is being created when the key is attached.



**Figure 5-1: tailored metaphor for key pairs**

The next pair of images in Figure 5-1 show secret and public keys as cursors, which will be used in context to signal that those types of keys should be applied to encrypted messages and to signed text, respectively. Finally, variations of the public key certificate image are created to visually signal that public keys are expired or retired, and, because

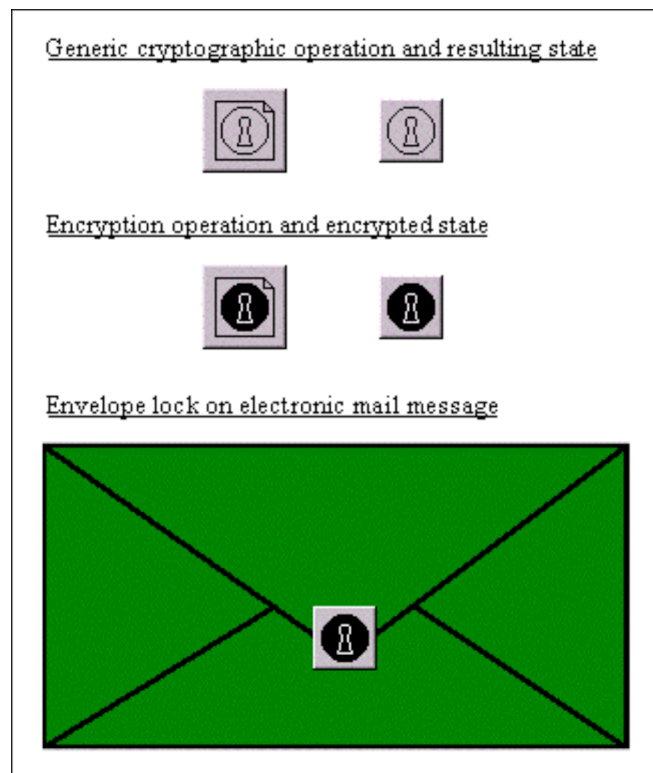


we thought users might find it useful, to support an additional state of being “blocked from use” at the user’s discretion.

Another important state for public keys is that of being signed or unsigned; that is not depicted here because it will be covered under the tailored metaphors for cryptographic operations.

## Tailored metaphors for cryptographic operations

Beginning at the top of the concept hierarchy, the qualities that need to be suggested for the generic cryptographic operation are that it provides security and that it requires a key. The top set of images in Figure 5-2 show the resulting tailored metaphor as a toolbar button, for the generic operation, and as a smaller, button-like object for the state that results from the operation. The style of the images continues the simple, cartoon-like theme established by the key pair tailored metaphor.



**Figure 5-2: tailored metaphor for encryption**

Next, the tailored metaphor for the generic operation is specialized for encryption, by making it black, which suggests both secrecy and a correct association with the secret key. In combination with the key cursors discussed in the previous section, this graphical image for the encrypted state can be used to strongly reinforce the user’s understanding that encrypted data is decrypted using a secret key.

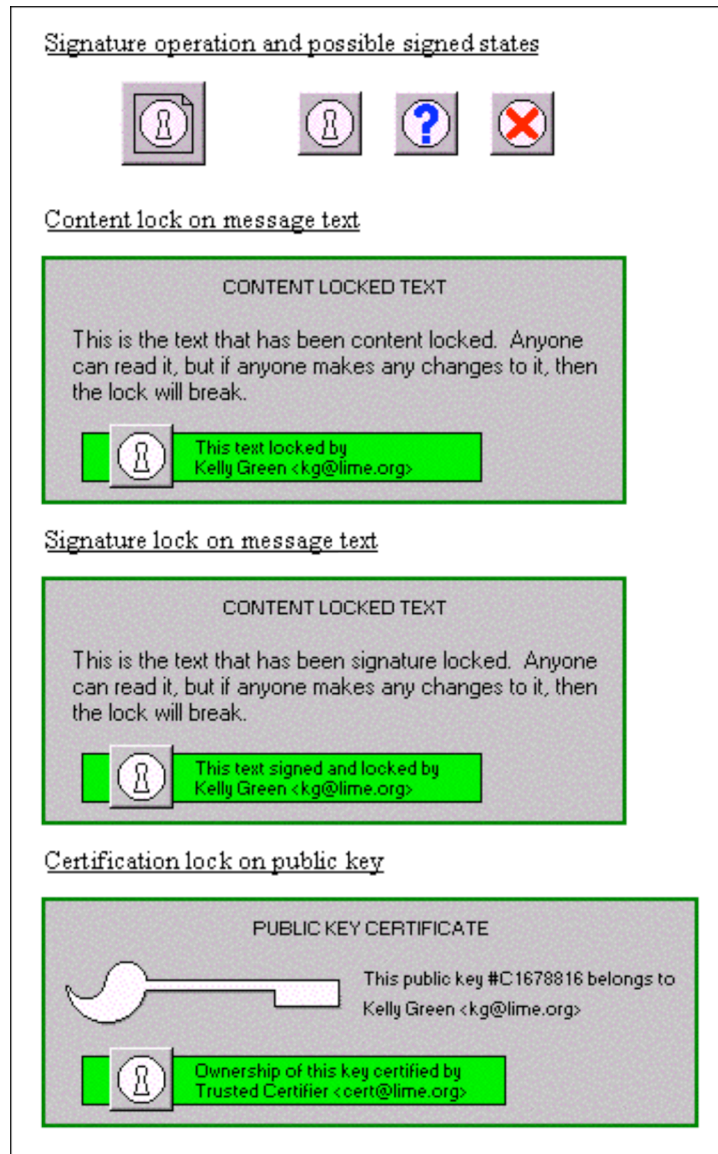
Finally, the tailored metaphor is specialized further to make both the purpose and the presence of encryption for email messages as clear and visual as possible, and the name “envelope lock” is coined for the resulting specialized conceptual object. When the user encrypts a message, the text disappears and is replaced by the envelope lock image, graphically demonstrating the information hiding that the encryption provides. To open the envelope lock and restore access to the text, the user must apply the appropriate secret key to the lock which is depicted as sealing the envelope closed. Moving the mouse over the image of the locked envelope causes the secret key cursor to appear, cueing the user to apply the key to open the lock.

The visual assertiveness of the envelope helps to ensure that the user will notice its absence as well as its presence, protecting against the accidental sending of private messages in plain text. The attractiveness of the image, suggesting the use of personal stationery, should encourage the user to view the use of encryption as a pleasant step in personal correspondence rather than an irritating burden.

Figure 5-3 shows the specialization from the tailored metaphor for the generic cryptographic operation to those for digital signatures. Note that, unlike most current visual metaphors for digital signatures, the use of the lock image as a base preserves the visual association with the use of keys. At the top level, the generic lock image is specialized by making it white, to suggest something that operates in the open, and that invites the application of a public key. The state image is further specialized to cover the cases of a signature for which no matching public key is available, indicated with a blue question mark, and of an invalid signature when the signed data has been tampered with, indicated by a red ‘X’.

For the three actual uses of digital signatures in the Lime electronic mail application, the white lock image is further elaborated to represent three variant conceptual objects: “content locks” and “signature locks” which are applied to text, and “certification locks” which are applied to public keys. All three depict the signed data as an uneditable graphical image, sealed within a gray rectangle that suggests laminating. The white lock image is applied to the bottom of the gray rectangle, suggesting a “personal seal”, and is shown as affixing a green ribbon on which is displayed the identifier associated with the public key that “fits the lock”, as well as a statement of the intent that the lock represents. This tailored metaphor thus illustrates all of the following:

- That a digital signature is a secure state/operation requiring a key.
- That a digital signature prevents modification of the data to which it is applied.
- That a digital signature identifies the owner of the key used to apply it.
- That a digital signature requires a matching public key for verification.
- That a digital signature breaks if the data to which it is applied is modified.
- That a digital signature may convey only that data is authentic, meaning tamperproofed, or it may also convey intent to supply a legal signature, or a testimonial as to the ownership of a public key.



**Figure 5-3: tailored metaphor for digital signatures**

## Conceptual model specification coverage

Table 5-2 shows the extended conceptual model specification again, this time with the items that are not addressed by the tailored metaphors indicated by italics. These will need to be given extra attention to ensure that they are well supported in the remainder of the design process.

|  |   |
|--|---|
| <p><u>objects</u></p> <p>Key pairs, each consisting of a secret key and a public key</p>   | <p><u>frustration risks</u></p> <p><i>Difficulty generating own key pair</i><br/> <i>Difficulty protecting own key pair with pass phrase</i><br/> Difficulty encrypting messages</p>  |
| <p><u>states</u></p> <p>Messages may be encrypted or plaintext.<br/> Message text may be signed or unsigned.<br/> Public keys may be signed or unsigned.<br/> Signatures may be valid, invalid, or unverifiable (when matching public key is not available).<br/> Public keys may be valid, revoked, or expired.</p>   | <p>Difficulty choosing correct public key with which to encrypt message<br/> <i>Difficulty getting other people's public keys</i><br/> Difficulty giving own public key to other people<br/> Difficulty signing messages<br/> Difficulty signing public keys<br/> <i>Difficulty getting own public key signed by an appropriate person</i><br/> Difficulty verifying signatures on messages or public keys<br/> <i>Difficulty revoking public keys</i></p>  |
| <p><u>operations</u></p> <p><i>Generation of key pairs</i><br/> <i>Protection of key pairs with pass phrase</i><br/> Encryption of plaintext messages<br/> Decryption of encrypted messages<br/> Signing of text<br/> Signing of public keys<br/> <i>Deleting signatures from text or public keys</i><br/> Verifying of signatures<br/> <i>Exporting public keys to files</i><br/> <i>Sending public keys to key servers</i><br/> Attaching public keys to messages<br/> <i>Importing public keys from files</i><br/> <i>Retrieving public keys from key servers</i><br/> <i>Importing public keys from messages</i><br/> <i>Deleting public or secret keys</i><br/> <i>Revoking public keys</i></p> | <p><u>bad citizen risks</u></p> <p><i>Ill-considered signing of other people's public keys</i></p>  |
| <p><u>accident risks</u></p> <p>Accidental exposure of secret key<br/> Accidental exposure of private message by emailing it without encryption<br/> Accidental acceptance of unsigned message as authentic<br/> Accidental acceptance of unauthenticated public key as authentic</p>  | <p><u>logical model concepts</u></p> <p>Everyone has their own key pair.<br/> Each person keeps the secret half of their key pair secret.<br/> Each person gives the public half of their key pair to everyone else.<br/> <i>Each person verifies the identity of the owner of each public key they are given.</i><br/> Any security operation performed using one half of a key pair can only be reversed by using the other half.<br/> Any security operation that can be reversed using one half of a key pair must have been originally performed using the other half.</p> |

**Table 2: Extended conceptual model specification, with gaps in tailored metaphor coverage indicated by italics**

## 5.4 Additional example: firewall port management

This chapter has illustrated metaphor tailoring using the example of Lime secure electronic mail. However, metaphor tailoring is a general technique useful for secure software. In this section, we illustrate metaphor tailoring with a completely different example: managing ports on a firewall.

Consumer oriented routers have become popular for supporting home networks with a broadband internet connection. One advantage of these types of routers is that they allow the user to specify security parameters and thus act as a simple firewall protecting the home network. Virtually all routers support some type of Network Address Translation (NAT) which allows a single broadband IP address to be multiplexed across the home network. Individual ports on home machines are dynamically mapped to unused ports on the router, effectively disrupting attacks which depend on access to particular ports. For example, port 23 is conventionally mapped to the *telnet* service, and offers outsiders an easy way to gain access to UNIX or Linux based machines. However, the NAT service will actually map port 23 to a random port, thus forcing the outsider to engage in a portscan to find the vulnerable port. To completely eliminate this vulnerability, consumer oriented routers offer further abilities to turn off access to particular ports. For example, a user may specify a rule saying that port 23 of machines inside the home network should never be mapped to an outside port unless the machine in the home network initiates the connection.

From the standpoint of user interfaces, these rulesets pose at least two issues. First, how can users effectively manage access to their ports? Second, if a user is using an application that demands particular ports, how can the user create a “demilitarized zone” that maps those ports to the machine running the application? (An example of such an application is Microsoft NetMeeting, which supports an interactive whiteboard but requires that users communicate over particular ports.) Conventional routers address these interface problems using a variety of techniques ranging from checkmarks in an array to formal rules written in a special purpose computer language. As an example of how complex these systems can become, a consumer router manufacturer, Netopia, publishes a router manual that is 258 pages long, and which devotes approximately 100 of those pages to specifying rules for router configuration.

The first step in metaphor tailoring is to create the enhanced conceptual model specification for the security system, with risk enumeration. For the sake of brevity, we present here a simplified, abbreviated risk enumeration:

**Accident risks** include: accidentally turning on or off the wrong ports, ignorance of port functions, failure to include correct ports in a demilitarized zone, and failure to understand functions associated with particular ports.

**Frustration risks** include: failure to be able to turn on appropriate ports to support a function, and, as a result, opening all ports on a given computer.

**Bad citizen risks** include: allowing rogue “zombie” programs on a given computer to send “denial of service” messages to remote machines as part of an attempt to force the remote machine off the internet, such as flooding a remote machine with SYN packets.

With this risk enumeration, we have in place the necessary requisite to begin metaphor tailoring. Here is one type of metaphor that a Security-HCI engineer might propose:

First, she might design a system where ports are described both by their commonly used Internet function, e.g. port 23 may say “telnet” together with a stoplight icon that indicates whether allowing access to that port is advised (green); potentially dangerous (yellow) or not advised (red). She might also propose a pictorial representation of the network, where ports that are mapped directly to machines through the demilitarized zone are visually attached to the representation of the machine on the diagram. A more elaborate scheme would allow ports associated with applications (such as NetMeeting) to be represented by a single icon which could then be attached to the representation of a machine in the visual presentation.

While this is a simple example, metaphor tailoring gave us a start in representing the example and the reader may enjoy completing the exercise of extending the metaphor to support additional functions.

## 5.5 Summary

In this chapter, we have presented an enhanced and specialized technique called *metaphor tailoring* for constructing highly effective named visual representations for security mechanisms, states, and operations. We have then demonstrated the application of that technique to create the set of visual metaphors for public key cryptography that are used in our Lime secure electronic mail application. Formal evaluation of the success of those metaphors will be presented in Chapter 6.

## CHAPTER 6 Experimental results

In this dissertation we have argued at the most general level that computer security requires its own theory and techniques for achieving usability, because it includes basic qualities and challenges that are fundamentally different from those of the applications that are the usual domain of human computer interaction research. We have then presented two techniques that we argue are generally applicable for addressing the usability needs of security: first, a reinvention of the technique of user interface staging, as a method for guiding users to an intuitive understanding of the purpose of those computer security mechanisms that would otherwise seem opaquely abstract and divorced from human concerns; and second, an explicit tailoring of visual metaphors to provide the particular kind of observability and predictability for which computer security demands the highest priority.

Continuing to move from the theoretical to the concrete, we have then demonstrated the use of both those techniques in the creation of a user interface design for electronic mail security using visible public key cryptography, and have named this design Lime. In this chapter we will discuss the evaluation of that design through two forms of user testing, the results of which provide strong support for the utility of both techniques.

The goals of the user testing were first to isolate and evaluate the success of the staging at making key certification understandable, then to do the same for the success of the visual metaphors at conveying the basic conceptual model of public key cryptography, and finally, to evaluate the success of the user interface design as a whole at reaching an acceptable usability threshold. Accordingly, the first two tests conducted were designed to separately compare the staging and the metaphors against several plausible design alternatives. These tests were done using paper presentations, and are discussed in sections 6.1 and 6.2. The final test was done using a full software implementation of the Lime user interface design, and is intended as a proof of concept; it is discussed in section 6.3. Finally, section 6.4 summarizes the results and our conclusions.

## 6.1 Staged user interface comparison

Since available time and resources did not permit the creation of multiple variant software programs, paper presentations were used to represent each user interface design to the test participants. These presentations covered briefly each of the basic concepts necessary to the use of public key cryptography, phrasing the explanation in terms of the conceptual model and visual metaphors of the variant being tested. Participant understanding was then evaluated via written answers to a series of printed questions.

### Objectives

The objective of this test was to evaluate the staged presentation of key authentication in my user interface design against its equivalent from two commonly seen user interfaces for public key cryptography, to test which best enabled users to understand key signing as a method for authenticating public keys.

### Test design and methodology

#### Presentation

In this test, each participant was given one variant of a 2-3 page printed presentation explaining a security program for use with electronic mail. In order to measure the effects of the variant presentations of key authentication in isolation from those of particular visual metaphors, no images were used in the presentations, and cryptographic objects and functions were referred to only in metaphor-neutral language: *tokens* rather than keys, *make-unreadable* rather than encrypt, and *make-tamperproof* rather than digitally sign. The presentations for each of the three variants were identical aside from the points described next. Copies of the three presentation variants are in Appendix B.

The three variants were:

**PGP:** the presentation explained the use of a token pair consisting of a public token and a secret token, the use of the functions make-unreadable and make-tamperproof, the importance of making sure you really have the right public token for each person, and certification as a mechanism for doing so. This variant has no staging.

**SSL:** as for PGP, but instead of explaining the token pair, the presentation explained that each person has a security token, and presented the use of the cryptographic functions according to that single token model. This corresponds to the way in which browsers present SSL by explaining that each person has a certificate. This variant also has no staging; it was included in order to evaluate a design that was simplified by hiding more information from the user.

**Lime:** as for PGP, but with the addition of two paragraphs discussing the use of social authentication methods, inserted before the explanation of certification. The



two additional paragraphs contained no references to key certification whatsoever, so that the presentation of key certification was identical across all three variants. This variant represents a design with a staged presentation of key authentication.

## Data collection

After the participant had read through his or her assigned presentation and indicated readiness to continue, they were given a series of five pages of printed questions about how they would use the security software in five different scenarios, and asked to answer in writing. The questions were identical for each variant except for slight changes in wording to match the SSL variant's discussion of security tokens rather than token pairs.

Each of the five pages of questions had the following format:

1. Brief scenario description.
2. Question asking whether it would be worth spending time to make the message secure, and if so, how much time.
3. Question asking which functions and tokens would need to be used.
4. Question asking how one would get the necessary tokens at an appropriate level of security.
5. Question inviting the participant to make comments.

After the five pages of questions were completed, the participant was given a short debriefing questionnaire, thanked, and paid \$10. Copies of the questions and of the debriefing questionnaire can be viewed in Appendix B.

## Data analysis

In analyzing the data, it was necessary to take into account the fact that the questions did not always make it clear to participants how much information their answers were supposed to provide. For example, when asked which tokens and which functions would need to be used, some participants answered with a list of tokens and a list of functions, without any clear association between the two. Such an answer could not be taken as evidence that the participant understood that one must use one's own secret token with make-tamperproof, or the intended recipient's public token with make-unreadable, but it was not necessarily indicative of lack of understanding either.

Due to this, our method of analyzing the collected data was to look for evidence of participant understanding of each important point, provided in response to any of the questions. Each page of questions thus represented an additional opportunity for the participant to demonstrate understanding. No participant was ever coded as having not understood a particular point, only as having either demonstrated or failed to demonstrate understanding.

Results were coded for demonstrated understanding of the following points (modified as appropriate for the SSL variant):

Encryption:

1. That make-readable/unreadable are the functions for privacy.
2. That the sender uses the recipient's public token and the recipient uses his or her own secret token.

Digital signatures:

1. That make/check-tamperproof are the functions for authentication.
2. That the sender uses his or her own secret token and the recipient uses the sender's public token.

Key trading and authentication:

1. That the sender and recipient need to have each other's public tokens.
2. That they must make sure those are the real public tokens for each person.
3. That a trusted third party can attest to the ownership of someone's public token for you, by using their own secret token to do make-tamperproof on it, if you already have the trusted third party's public token.

A second coder then independently coded results for six randomly selected test participants; 89% of those results matched exactly, and the remainder were within plus or minus one value. A collation of the raw data and copies of the coding sheets used can be viewed in Appendix B. The data from the debriefing questionnaire was not coded as it did not appear to hold any significant information.

## Participant selection

Participants were recruited by putting up posters on the Carnegie Mellon University campus, and by posting an ad to the newsgroups `cmu.misc.market` and `pgh.general`. The posters and ad offered payment of \$10 for one hour of participation in a research project for improving computer security.

Respondents to the recruitment were asked to answer a set of intake questions in order to establish basic demographic information and to screen out those who already understood the basics of public key cryptography (see Appendix B). Those who passed the screening were randomly assigned to one of the three design variants for testing. The random assignment was accomplished by assembling all the test materials needed for 30 participants, 10 for each of the three variants to be tested<sup>7</sup>, and placing them into 30 unmarked folders, which were then thoroughly shuffled. When each participant arrived for their individual test session, they were given the test materials from the next folder at the top of the stack, so that the monitor, conducting the test, did not know which of the

---

<sup>7</sup> By accident, the set of folders assembled actually contained 11 instances of the Lime variant, 10 instances of the PGP variant, and 9 instances of the SSL variant. The results are thus analyzed and presented in terms of percentages.

three variants the participant would be receiving until after the assignment had been made and the test session had begun.

## Results

Our hypothesis predicted that the Lime variant would enable more test participants to understand key certification, due to its staged design in which participants were first introduced to key authentication in terms of weaker, social methods. The test results supported this, as shown in Table 6-1: 45% of the test participants who received the Lime variant demonstrated understanding that public keys need to be traded and authenticated, and correctly described key certification as a method for doing so, versus 10% of the test participants who received the PGP variant, and none of the participants who received the SSL variant. Insofar as this printed presentation is an accurate stand-in for an actual software application, this is a strong result in favor of staging.

| <b>Points correctly described by participant</b>      | <b>Variants</b> |            |            |
|---|-----------------|------------|------------|
|   | <b>Lime</b>     | <b>PGP</b> | <b>SSL</b> |
| key trading, authentication, and certification        | 45%             | 10%        | 0%         |
| key trading and authentication, but not certification | 36%             | 50%        | 89%        |
| key trading, but not authentication or certification  | 18%             | 40%        | 0%         |
| not even key trading                                  | 0%              | 0%         | 11%        |

**Table 6-1: Participant understanding of key certification in staging test**

Another interesting result from Table 6-1 is that, if we disregard certification, and look just at participant understanding that public keys must be traded and authenticated, then the SSL variant yielded quite good results. Our impression is that this is because the participants who received that variant tended to think of security tokens as passwords, which led them to make reasonably correct assumptions about trading tokens and making sure one has the right token, but did not help them at all with the concept of certification.

| <b>Points correctly described by participant</b> | <b>Variants</b> |            |            |
|--|-----------------|------------|------------|
|  | <b>Lime</b>     | <b>PGP</b> | <b>SSL</b> |
| use for privacy, plus which keys are needed      | 73%             | 80%        | 56%        |
| use for privacy, but not which keys are needed   | 27%             | 20%        | 44%        |
| not even that encryption is for privacy          | 0%              | 0%         | 0%         |

**Table 6-2: Participant understanding of encryption in staging test**

| Points correctly described by participant               | Variants |     |     |
|---|----------|-----|-----|
|   | Lime     | PGP | SSL |
| use for authentication, plus which keys are needed      | 64%      | 70% | 56% |
| use for authentication, but not which keys are needed   | 36%      | 20% | 44% |
| not even that digital signatures are for authentication | 0%       | 10% | 0%  |

**Table 6-3: Participant understanding of digital signatures in staging test**

In looking at the results for participant understanding of encryption (Table 6-2) and digital signatures (Table 6-3), an important question is whether the gain in understanding key certification that we see for the Lime variant comes at a cost of less understanding of other points, perhaps due to overwhelming the participant with extra information. Fortunately, this does not appear to be the case. The results for the Lime and PGP variants are basically equivalent; the SSL variant results are basically equivalent for digital signatures and slightly worse for encryption.

In summary, the results of this test suggest that there is significant benefit to staging the presentation of key certification to the user, and little or no associated cost. They further suggest that attempts to simplify by concealing the basic key pair model will not improve user understanding of how to encrypt and digitally sign, and may even harm it somewhat.

## 6.2 Visual metaphor set comparison

This test again used variant paper presentations, very similar to those used in the staging comparison test, but this time presenting the participants with visual metaphors.

### Objectives

The objective of this test was to compare three different sets of visual metaphors for public key cryptography, to see which best enabled test participants to understand the use of encryption, digital signatures, and key management.

### Test design and methodology

#### Presentation

The design of this test was very similar to that of the previous test. Each participant was given one variant of a 2-3 page printed presentation explaining a security program for use with electronic mail. The presentation text was taken from the Lime variant presentation of the previous test, so that all three metaphor set variants for this test were accompanied

by text that presented explicit key pairs and staged key authentication. Aside from necessarily metaphor-specific terminology, such as references to encrypting versus to applying an envelope lock, that text was identical for all three metaphor set variants. Copies of the three presentation variants are in Appendix C.

The three metaphor set variants were:

**PGP:** the brass keys, envelope/padlock, quill pen, and ASCII representations taken directly from PGP 5.0.

**Basic Lime:** the yin-yang key pair, envelope locks, and content locks.

**Extended Lime:** the yin-yang key pair, envelope locks, content locks, signature locks and certification locks.

## Data collection

Again, the data collection performed was very similar to that of the previous test. After reading through the assigned presentation, the participant was given a series of six pages of printed questions about how they would use the security software in six different scenarios, and asked to answer in writing. Five of the six pages were, aside from metaphor-specific terminology, identical to those used in the previous test; the sixth was added in order to include a scenario that would require the participant to describe a way to indicate a legal signature.

After the six pages of questions were completed, the participant was given a short debriefing questionnaire, thanked, and paid \$10. Copies of the questions and of the debriefing questionnaire can be viewed in Appendix C.

## Data analysis

Analysis of the collected data and coding of results was performed exactly as for the previous test, except for one additional coded point, which was whether the participant described a valid method for distinguishing an intended legal signature from basic tamperproofing. Second coder results for six randomly selected participants matched at 92%.

## Participant selection

Participants were recruited and selected exactly as described for the previous test, using the same advertisements, screening process, and randomized assignment to variants. Several data sets had to be destroyed due to our having forgotten to screen out minors, whom the terms of my human subjects authorization did not permit us to test. The results

presented next are thus based on a testing set of 26 participants: 8 who received the PGP variant, 9 who received the basic Lime variant, and 9 who received the extended Lime variant.

## Results

Tables 6-4, 6-5, and 6-6 show the results of this test. They do not support our hypothesis, as the PGP visual metaphors yielded better results than either of the Lime metaphor sets on all points except key certification and indication of a legal signature. However, there is something else interesting here: with the single exception of the PGP variant results for encryption, all of the results for all three variants are substantially worse than the results for the staged variant in the previous test. Since there were only three differences between this test and the test of that variant, it appears that one or more of those differences had a negative effect on participant understanding. Those differences were:

1. The addition of one more page of questions, with a scenario designed to require the indication of intent to provide a legal signature. Since this page of questions was given to the participants last, it could not have affected their answers to the earlier questions, and thus cannot have caused the difference in participant understanding.
2. The addition of pictures, representing visual metaphors for the cryptographic objects and functions, to the variant presentations. It is possible that this caused the participants to be distracted or confused and thus contributed to their achieving lower levels of understanding than those who were given a text-only presentation.
3. The change from the metaphor-neutral terms, such as *make-unreadable*, to the terms of the variant metaphor sets, such as *encrypt* or *envelope lock*. When we consider this change alongside the fact that the single improved result was that of the PGP variant for encryption, it suggests that the results of this test can be explained as corresponding to the familiarity of the test participants with the terminology used in the presentations. That is, in the test of the PGP metaphor variant, it is very likely that most if not all of the participants had previous exposure to the term *encryption* and knew basically what it is for, even though they did not know how public key cryptography works. In the previous test of the staged, metaphor-neutral variant, the terms used were very literally informative: *make-unreadable* and *make-tamperproof*. It makes sense that the change to less literal, more unfamiliar terminology would cause participant understanding to worsen.

| <b>Points correctly described by participant</b>      | <b>Variants</b> |                   |                 |
|---|-----------------|-------------------|-----------------|
|   | <b>PGP</b>      | <b>Lime basic</b> | <b>Lime ext</b> |
| key trading, authentication, and certification        | 0%              | 11%               | 22%             |
| key trading and authentication, but not certification | 88%             | 67%               | 56%             |
| key trading, but not authentication or certification  | 13%             | 22%               | 11%             |
| not even key trading                                  | 0%              | 0%                | 11%             |

**Table 6-4: Participant understanding of key certification in metaphor test**

| <b>Points correctly described by participant</b> | <b>Variants</b> |                   |                 |
|--|-----------------|-------------------|-----------------|
|  | <b>PGP</b>      | <b>Lime basic</b> | <b>Lime ext</b> |
| use for privacy, plus which keys are needed      | 88%             | 44%               | 44%             |
| use for privacy, but not which keys are needed   | 13%             | 44%               | 33%             |
| not even that encryption is for privacy          | 0%              | 11%               | 22%             |

**Table 6-5: Participant understanding of encryption in metaphor test**

| <b>Points correctly described by participant</b>        | <b>Variants</b> |                   |                 |
|---|-----------------|-------------------|-----------------|
|   | <b>PGP</b>      | <b>Lime basic</b> | <b>Lime ext</b> |
| use for authentication, plus which keys are needed      | 50%             | 33%               | 33%             |
| use for authentication, but not which keys are needed   | 50%             | 56%               | 67%             |
| not even that digital signatures are for authentication | 0%              | 11%               | 0%              |
| also indicating intent as legal signature               | 13%             | 11%               | 89%             |

**Table 6-6: Participant understanding of digital signatures in metaphor test**

Our main conclusion here is that this was not a successful test design: it did not do a good job of measuring the relative value of the different visual metaphor sets. A better test design would have changed only the pictures and not the terminology, in order to isolate the usefulness of the visual metaphors and prevent the test from reducing to familiarity with their labels. An ideal test design would have incorporated the terminology changes, but would also have given the participants the experience of interacting with the visual metaphors in some way equivalent to that which occurs in a real software program.

We do not conclude from these results that the best design solution would be one which used the metaphor-neutral, literally informative terminology from the previous test. If we were constrained to design a text-only interface to a public key cryptography system, then that terminology might well be ideal. In a graphical user interface, however, metaphors may allow us to provide users with visual objects that are directly manipulable in ways that significantly support user understanding, and it is not clear whether, in that context, the most usable names for those metaphors are those that correspond to the appearance of the object or those that correspond to its function. We therefore chose to proceed with

the implementation of our extended metaphor set in the Lime software, and to see how the test participants interacted with it in the proof of concept test described in the next section.

Finally, there was one additional minor result from this test, regarding the advanced uses of digital signatures. When provided with the metaphor of signature locks for use in indicating a legal signature rather than just tamperproofing, nearly all of the test participants correctly described the use of a signature lock when given the appropriate scenario and questions. Very few of the participants who received the PGP variant or basic Lime variant were able to come up with and describe a method for securely specifying a legal signature. This is not a surprising result, but it does support the hypothesis that the distinction between content and signature locks makes sense to people and would be practically useful. The Lime metaphor variants also appeared to have a small edge over the PGP variant in participant understanding of key certification, but the percentage differences are too small to be clearly significant.

## 6.3 Proof of concept user testing

In order to conduct the next phase of testing, we used Visual C++ to implement a complete simulation of our hypothetical Lime electronic mail security program. This simulation, which we will refer to as Lime, resembled a basic version of the Netscape or Mozilla mail client, but presented public key cryptography using our extended visual metaphor set and staged introduction to key certification. Network communication protocols were included to allow Lime to connect to the Limeserver, another small software program that ran on a UNIX workstation and provided mail server and key server functionality during the user tests.

It must be acknowledged that, since Lime is a complete user interface, its presentation of public key cryptography necessarily includes some other user interface design elements beyond those that directly relate to my staging and metaphor hypotheses. These other elements are neither extensive nor unusual, however, and thus are unlikely to have much impact on participant success. We also wish to stress here that this implementation of Lime is not intended to represent the best possible attempt at making public key electronic mail security usable. Such an attempt would at minimum incorporate good warning messages and a more extensive help system, and would probably benefit from various methods of scaffolding, such as those we will discuss in our chapter on future work.

In discussing this user test, we will use the terminology that corresponds to the Lime metaphors, i.e. envelope locks rather than encryption, and content locks and signature locks rather than digital signatures.



## Objectives

The objectives of this user test were:

1. To observe the test participants' interaction with the staged introduction to key certification, to see whether they made use of social authentication methods when given a scenario where weak security was appropriate, and to see whether they then successfully made use of key certification when given a scenario calling for stronger security.
2. To observe the test participants' interaction with the visual metaphors used to present public key cryptography, and in particular, to see whether they were successful at using the right public keys to create envelope locks and at using envelope, content and signature locks correctly when it was appropriate to do so.
3. To observe the test participants' overall level of success at using public key cryptography with the Lime user interface design, and to identify remaining usability problems and possible future solutions.

These observations will be the basis on which we assess whether the Lime user interface design reaches a usability threshold that is sufficient to enable most electronic mail users to successfully make use of public key encryption, digital signatures, and decentralized key certification.

## Test design and methodology

### Basic test format

This test was based on the PGP usability test discussed in Chapter 3, but was more extensive, incorporating three scenarios rather than one, and with a time limit of three hours rather than 90 minutes<sup>8</sup>. Twelve participants were tested<sup>9</sup>.

Test sessions were conducted in a small private lab in Soda Hall on the U.C. Berkeley campus. The participant was seated at a networked computer running a fresh, uninitialized copy of Lime, and we, as the test monitor, were seated approximately four

---

<sup>8</sup> The time allotted for the test itself was actually closer to two hours and thirty minutes, because the three hours allotted included briefing and debriefing. The testing began as soon as briefing was accomplished, and continued until all scenarios were complete or until there were only fifteen minutes remaining for debriefing, whichever occurred first. This is in contrast to the PGP test, which continued until the scenario was complete or until 90 minutes had elapsed, measured from the end of the initial briefing.

<sup>9</sup> There were also two discarded test sessions: those of participants P2 and P3. P2 did not cooperate with the test scenarios, treating the test session instead as a software quality control exercise and focusing on design issues such as button layout. P3 repeatedly triggered a previously unidentified software bug that caused Lime to crash whenever a content lock was used; after several occurrences of this we terminated the test session.

feet behind them and to the left, positioned to observe their actions on the screen. Participants were requested to think aloud as much as possible.

We used a laptop both to take detailed notes on the test session and to play the roles of the various scenario characters, using our own copy of Lime to connect to the Limeserver and to send and receive messages. The test session was also recorded by a videocamera positioned on a table behind the participant.

## Scenario and questionnaire design

The design of the test itself was based on the PGP usability test discussed in Chapter 3, but with a number of important modifications and elaborations. Instead of the single scenario in the PGP test, the Lime test used a sequence of three scenarios:

1. An initial, low security scenario in which the participant was told that they had acquired the Lime email security software and should try to use it to send a secure message to an old friend, “Steve”. The scenario description provided enough background information about Steve for the participants to attempt social authentication, if they chose to do so.
2. A higher security scenario similar to the main scenario in the PGP test, in which the participant was told that they were volunteering for a political campaign and had been given the duty of securely disseminating information to the campaign team members, beginning with the provided secret memo. In the Lime test, however, this scenario was elaborated to include the following events:
  - a. At the start of the scenario, the participant was given a floppy disk containing the public key of the campaign manager, “Maria”. The scenario description stated that Maria had exchanged public keys with the participant, in person and using floppy disks, at the time that the participant was hired.
  - b. If the participant sent the secret memo to the campaign team members without using a content or signature lock, a campaign team member responded with a complaint that they “couldn't be sure that message really came from you.”
  - c. If the participant sent the secret memo to the campaign team members with a content or signature lock, but had not provided the team members with a certified copy of their public key, then a campaign team member responded with a complaint that they “couldn't be sure that was really your public key.”
  - d. If the participant sent email to Maria requesting that she certify their public key, Maria complied.

- e. When the scenario was otherwise complete, the participant was sent an email purporting to come from one of the campaign team members, offering a new, uncertified public key and saying “I've made myself a new key, please use this one from now on.” This will be referred to as the “Attacker-Sam” message in the discussion of results to follow. Its purpose was to provide an additional opportunity for the participant to require that the new key be certified.
  - f. In the first test sessions, this scenario ended with a message from Maria saying that a hacker had broken into the office machines and that everyone needed to retire their key pairs and make new ones. In some sessions this created problems when participants reacted by deleting their copies of the team members' public keys, which were needed for the third scenario. Due to this, the retirement subscenario was moved to the very end of the test for the remaining test sessions.
3. An addendum to the previous scenario, in which the participant was told that their volunteer duties would now include the processing of requisitions from the campaign team members, and was given a brief set of rules for rejecting or approving those requisitions. Requisitions were to be rejected if they did not carry the submitting team member's signature lock, and the participant was told to approve acceptable requisitions by adding their own signature lock.

After the test scenarios were completed, or in some cases halted as the allotted time expired, participants were asked to fill out a debriefing questionnaire that contained the following sections:

1. A series of fourteen randomized statements about the correct use of the security functions in Lime, to be marked true or false.
2. A series of hypothetical security scenarios, for each of which the participant was asked to state how important security would be to them and whether they thought they would be able to use Lime to get the desired level of security with a reasonable amount of effort.
3. A request for the participant to add any comments that they would like to make.

## Data collection

The raw data collected consisted of the following:

1. The detailed notes we made while observing each test session.

2. The email messages sent and received by the participants and by us in the roles of the various scenario characters.
3. The videotapes of the test sessions.
4. Answers to the debriefing questionnaire given to the participants at the end of the test session.

## Data analysis

The raw data was processed into results using the coding book of boolean and multiple choice questions shown in Appendix D. A second coder also independently processed the data from two randomly selected test sessions for comparison, showing an exact match rate of 84%.

## Participant selection

Participants were recruited by posting an ad to the local community web site Craig's List<sup>10</sup>, offering \$10/hour for 2-3 hours of participation in a research study to improve computer security. Respondents were screened using an intake questionnaire, and those who had prior experience with PGP or could roughly explain the difference between public and private key cryptography were excluded.

## A note regarding comparison to the PGP case study

Although the design and goals of this user test are very similar to those of the PGP case study discussed in Chapter 3, it is important to stress that the results of the Lime and PGP tests cannot validly be held to direct comparison. The PGP test results demonstrated that significant usability problems existed for the PGP 5.0 user interface at that time. The Lime test results demonstrate that, with the Lime user interface at the present time, public key cryptography for electronic mail is usable for most security novices. Neither set of results rules out the possibility that the user population has changed enough that even security novices would now be equally successful with the PGP user interface; however, there is no anecdotal evidence suggesting that to be the case.

---

<sup>10</sup> <http://www.craigslist.org>

## Results

Our discussion of the results will be separated into those which relate directly to key authentication and certification and thus are relevant to the staged components of our design, and those which relate to use of the basic functions of public key cryptography and are expected to reflect the usability of the visual metaphors. Following that we will then also discuss individually those participants who encountered serious difficulties, and touch briefly on potential design solutions that could have averted those problems.

### Key certification results

In this section we present and discuss those test results that are relevant to the staged presentation of key certification in my user interface design. Those results are primarily drawn from participant actions in the second, higher security test scenario, and record the following:

- Whether the participant successfully authenticated their own public key by first getting Maria to certify it and then sending the certified key to the team members.
- Whether the participant required the campaign team members to provide certified copies of their public keys, first during the exchange of public keys preparatory to sending out the envelope locked secret memo, and second, upon receipt of the new uncertified key sent by Attacker-Sam.
- Whether the participant certified any keys inappropriately, or requested certification of their own public key from team members who could not validly comply.
- Participant scores on the true/false questions in the debriefing questionnaire that relate to key certification.

Because our hypothesis is that the staging has a significant positive effect on participant success at those tasks, we present the results with the participants grouped according to the best available measure of their interaction with that staging, which is whether or not they made use of social authentication when exchanging public keys with Steve in the first scenario.

In these results only nine of the twelve test participants had the opportunity to fully attempt the certification tasks. The three who did not were P7, P10, and P11, who for that reason are shown marked with an asterisk in the tables that follow. P7 and P10 did not succeed at the prerequisite task of trading public keys with the campaign team

members<sup>11</sup>, and P11 misunderstood a directive about the scenario description as telling her to ignore all key authentication issues<sup>12</sup>.

## Results for authenticating own public key

Table 6-7 shows the results for whether the participant got Maria to certify their own public key. The surprising success of P10 here appears to have been accidental, as he requested certification for his key while trying to figure out why the team members couldn't open his envelope locked messages.

|                                  | Used social auth     | No social auth | Total |
|----------------------------------|----------------------|----------------|-------|
| <b>Immediate success</b>         | -                    | P10*           | 1     |
| <b>Success after complaint</b>   | P4, P6, P9, P12, P13 | P5, P14        | 7     |
| <b>Success after direction</b>   | P1                   | -              | 1     |
| <b>Failure or not applicable</b> | P7*, P8, P11*        | -              | 3     |

**Table 6-7: Participant success at getting own public key certified**

More promising are the results for the participants listed as “Success after complaint.” These participants initially sent their public key to the team members without certification, and then received a complaint in response saying that the team member “needed a way to make sure that key really came from you,” after which they successfully got the key certified by Maria. The single participant listed as “Success after direction” did the same, except that the complaint he received said “you should have Maria certify your key.”<sup>13</sup>

P7 and P11 failed for the reasons described earlier. The other failure, P8, clearly understood that certification was needed, but not how to validly acquire it, as he repeatedly responded to key authentication complaints by certifying and sending out copies of his own public key and, when that did not work, of the other public keys in his possession.

<sup>11</sup> The reasons for their difficulty will be discussed later in this chapter. Both did successfully trade public keys with Steve in the first scenario.

<sup>12</sup> Several participants began the second scenario by attempting to make up some shared personal data that they could use for social authentication with the campaign team members, such as the color of the shirt they were wearing on the day they were hired, or the correct pronunciation of their last name. For those who did this, we verbally clarified the scenario by telling them that they had never met the campaign team members in person or talked to them before. Participant P11 appeared to interpret our direction as meaning that key authentication issues were to be disregarded for that scenario: she ignored all authentication related tasks and complaints from the campaign team from that point on. It did not appear to us that there was any good way to correct the misunderstanding without inappropriately stressing the importance of authentication, so we did not try to do so.

<sup>13</sup> This was an error on our part, as our intent was to have the team members only make complaints in terms of the desired end goal, such as “I need a way to make sure this is really your key,” and to avoid any reference to the tasks required to achieve that goal. There were no other instances of this error.

Next, Table 6-8 shows the results for whether the participant successfully provided the certified copy of their public key to the campaign team members. The six participants in the top two rows are considered to have succeeded, since they sent their certified keys either to the whole team or at least to the team member who made the key authentication complaint.

|                            | <b>Used social auth</b> | <b>No social auth</b> | <b>Total</b> |
|----------------------------|-------------------------|-----------------------|--------------|
| <b>Sent to all</b>         | P1, P6, P13             | -                     | 3            |
| <b>Sent to some</b>        | P9, P12                 | P14                   | 3            |
| <b>Tried to send</b>       | P4                      | -                     | 1            |
| <b>Did not try to send</b> | -                       | P5, P10*              | 2            |
| <b>Not applicable</b>      | P7*, P8, P11*           | -                     | 3            |

**Table 6-8: Participant success at sending certified copy of own public key to team members**

P4, who is listed under “Tried to send”, had some difficulty getting his certified key imported from Maria’s response and pasted into his outgoing message; he resolved that problem quickly, but then, before sending out the certified key, began rechecking the certification locks on all the team members’ keys and never returned to the task of sending out his own.

P5 and P10 got Maria to certify their keys, but then never tried to pass on the certified key to the campaign team members. P5 appeared to think that when Maria certified his key the certification somehow took effect globally, since upon receiving her reply he immediately sent out the envelope locked secret memo again, but without attaching a copy of his certified key. When he then received another key authentication complaint, he expressed worry that there was something wrong with the certification, or that he had accidentally used the wrong key. P10 never attempted to trade public keys with the campaign team members at all. Lastly, the participants listed under “Not applicable” are those who did not succeed in getting their own public keys certified, as seen in the previous table.

These are very promising results. In only one case, that of P8, did we observe a real failure to understand how to validly get certification for one’s own public key. P4’s failure to send the certified key to the team members appeared to be a lapse of attention and not of understanding. P5’s failure to realize that he needed to send out his key again to publicize the certification represents a problem, but one that should be easy to address by adding reinforcement at appropriate points, such as when a newly acquired certification lock is imported into the Key Pair Vault. Since all but one of the participants for whom this part of the scenario was applicable were able to get their own public keys certified, and most then successfully provided their certified public keys to the complaining team member, the Lime design appears to meet the target usability threshold with regard to use of certification to authenticate one’s own public key.

## Results for requiring authenticated public keys from others

Table 6-9 shows the results for whether the participant withheld the secret memo until the team members provided certified copies of their public keys. Only P8 and P9 did so, and each under special circumstances. P8 chose the creative but acceptable solution of envelope locking and sending the secret memo to the certified key owners only, with a request that they pass on the information to the remaining team member. As for P9, the special circumstance is that he received an authentication complaint about his own public key from a team member before he had sent out any copies of the secret memo, and thus had that example to reinforce the idea that authenticating keys was to be treated as important.

The majority of the participants envelope locked the secret memo with the public keys as they were sent by the team members, without complaining about the uncertified key. P7 and P10 did not trade public keys with the team members and thus never reached this task.

|                            | Used social auth           | No social auth | Total |
|----------------------------|----------------------------|----------------|-------|
| <b>Certified keys only</b> | P8, P9                     | -              | 2     |
| <b>Uncertified keys</b>    | P1, P4, P6, P11*, P12, P13 | P5, P14        | 8     |
| <b>Not applicable</b>      | P7*                        | P10*           | 2     |

**Table 6-9: Participant insistence on certified public keys for putting envelope lock on secret memo**

A participant who understood that, ideally, public keys should be certified might still be unsure as to whether they were meant to aggressively insist that the team members provide certified keys or to passively accept the keys they were given. To take this into account, the test scenario gave participants a second chance, by having Attacker-Sam send a new, uncertified public key after the participant received and resolved any key authentication complaints from the team members. Table 6-10 shows participant reactions to receiving Attacker-Sam's uncertified public key.

|                               | Used social auth         | No social auth | Total |
|-------------------------------|--------------------------|----------------|-------|
| <b>Demanded certification</b> | P1, P4, P6, P9, P12, P13 | -              | 6     |
| <b>Accepted uncertified</b>   | P11*                     | P14            | 2     |
| <b>Unclear reaction</b>       | -                        | P5             | 1     |
| <b>Not applicable</b>         | P7*, P8                  | P10*           | 3     |

**Table 6-10: Participant insistence on certification for new public key sent by Attacker-Sam**

Now we have six participants responding to Attacker-Sam with a request for key certification. As for the others, P14 imported Attacker-Sam's key and appeared to accept it as valid. P11 responded with an apparently unserious "this is you, right?" message with a smiley face, and imported the new key. P5, who appeared to be fairly tired and frustrated by this point in the session, responded with a message saying "what hardware



store did you go to, to make your new key?” and is thus listed above under “Unclear reaction.” P7, P8 and P10 did not receive the message from Attacker-Sam.

Ideally, we would like to see all participants proactively require validly certified public keys from the start, but that may never be realistic in a test that asks participants to use their own judgement within a social situation. What we do see here is that when participants were given the small social reinforcement of having a team member complain that a public key was not authenticated, and then were offered an unauthenticated public key, a strong majority responded by requiring certification. This suggests that, in real world situations, most users would be able to require and evaluate key certification once they decide that strongly authenticated keys are called for.

### Results for inappropriate use of key certification

Another useful measure of participant understanding of key certification can be taken from whether they also used it in inappropriate ways. The first three rows of Table 6-11 show the participants who certified other people’s public keys when they had not met with those people to exchange keys in person, and the last row shows those who asked for certification for their own public key from people other than Maria, who was the only person who could validly provide it.

|  | Used social auth  | No social auth | Total |
|--|-------------------|----------------|-------|
| <b>Certified Steve’s public key in first scenario</b>  | P7, P8, P9        | P10, P14       | 5     |
| <b>Certified campaign team member’s otherwise uncertified public key</b>                       | P11*              | P14            | 2     |
| <b>Certified campaign team member’s public key that had previously been certified by Maria</b> | P8, P9, P11*, P13 | -              | 4     |
| <b>Requested certification for own public key from someone other than Maria</b>                | -                 | P10            | 1     |

**Table 6-11: Inappropriate use of key certification by participants**

While quite a few participants did inappropriately certify other people’s public keys, none of them mailed out or otherwise publicized the resulting certification. By contrast, participants who took the valid step of certifying Maria’s public key often sent a copy to her afterwards. Because of this, we do not consider the inappropriate certifications to be a serious problem, but do consider this an appropriate area for added warning messages. As for P10, he sent an email message to the entire team, including Maria, asking for his key to be certified, so it is unclear whether his intent was really to ask each team member

to provide the certification, or just to generally signal that he needed help getting it. We therefore do not consider his action to represent a serious problem either.

### Certification related scores on debriefing questionnaire

Table 6-12 shows the results for the three true/false statements in the debriefing questionnaire that referred to key authentication and certification. All participants agreed that “You can trade public keys by email and be fairly sure that you are getting the right public key, as long as both people include personal messages that talk about things a stranger wouldn’t know.” Ten participants correctly disagreed that “You can tell whether you have the right public key for someone by checking the name and email address attached to it,” and ten correctly disagreed that “Putting your certification lock on a public key means that you are the owner of that public key.” These are very positive results, as we would not expect the participants to have understood those points prior to the test session. We note also that the four who missed a question are all participants who had some trouble with the certification related tasks.

|                     | <b>Used social auth</b>      | <b>No social auth</b> |
|---------------------|------------------------------|-----------------------|
| <b>3/3 correct</b>  | P1, P4, P6, P7, P9, P12, P13 | P5                    |
| <b>2/3 correct</b>  | P8, P11                      | P10, P14              |
| <b>1/3 correct</b>  | -                            | -                     |
| <b>none correct</b> | -                            | -                     |

**Table 6-12: Participant scores on certification related questions in debriefing test**

### Results for use of basic cryptographic functions

In this section we present and discuss those test results that are relevant to the visual metaphors used to present the cryptographic objects and functions to the user. These results record the following:

- Whether the participant succeeded at the “primary task” of sending the secret memo out to all the campaign team members, correctly content locked and envelope locked with all the correct public keys.
- Whether the participant successfully avoided ever sending anything private without an envelope lock.
- Whether the participant successfully avoided the error of using one’s own public key to envelope lock a message meant for others, and if not, how quickly the participant was able to recover.

- Whether the participant always used a correctly applied content lock when sending out the secret memo.
- Whether the participant successfully evaluated and made use of signature locks in the requisition scenario.

## Results for primary task

Table 6-13 shows the results for participant success at sending out the secret memo to the campaign team with a correctly applied content lock and envelope locked with all the correct public keys. A solid majority of the participants achieved complete success for this task. Of the others, P9 used all the correct keys to envelope lock, but content locked only her name at the end of the message, and not the content of the secret memo itself. P11 used all the correct keys to envelope lock, but did not use a content lock. P7 and P10 did not succeed in trading public keys with the campaign team, and thus could not complete this task.

|  | <b>Participants</b>               |
|--|-----------------------------------|
| <b>Used correct envelope and content locks</b>         | P1, P4, P5, P6, P8, P12, P13, P14 |
| <b>Used correct envelope lock but bad content lock</b> | P9                                |
| <b>Used correct envelope lock but no content lock</b>  | P11                               |
| <b>Sent without envelope lock or did not send</b>      | P7, P10                           |

**Table 6-13: Participant final success at securely sending secret memo to team**

Given that every participant who successfully traded public keys was also able to successfully send out the secret memo to all the team members with a correct envelope lock, and that almost all of those also used a correct content lock, the results for this task certainly meet the usability threshold previously specified.

## Other results for use of envelope locks

Table 6-14 shows the results for participant success at avoiding exposure of private information. Nine of the twelve participants always used an envelope lock on all private messages. Of the other three, P7 sent a message in the first scenario that might have been intended as private, without using an envelope lock, but always used an envelope lock when sending the secret memo. Only P10 and P14 ever sent the secret memo without an envelope lock.

|  | <b>Participants</b>                   |
|--|---------------------------------------|
| <b>Always envelope locked private messages</b> | P1, P4, P5, P6, P8, P9, P11, P12, P13 |
| <b>Always envelope locked secret memo</b>      | P7                                    |
| <b>Sent secret memo without envelope lock</b>  | P10, P14                              |

**Table 6-14: Participant success at protecting secrets**

Both of the participants who sent the secret memo without an envelope lock appeared to be doing so in the belief that use of a different cryptographic object had provided encryption. In the case of P10, it was attaching a public key to the message, and in the case of P14, it was use of a content lock. Both of these potential misunderstandings may thus need to be more explicitly addressed. However, the high percentage of participants who avoided ever making this error supports the conclusion that the envelope lock metaphor is a good one.

Table 6-15 shows the results for participant success at avoiding the standard error of using one's own public key to envelope lock messages meant for others. Five participants avoided this error entirely and always used the correct public keys to encrypt. Another five did initially make this error, but recovered quickly after receiving a reply stating only that the recipient "can't open that last message you sent," and used the correct public keys from then on. The remaining two are P7 and P10, who never got the correct public keys at all.

|   | <b>Participants</b>  |
|---|----------------------|
| <b>Always used the correct public keys</b>                | P4, P5, P6, P12, P13 |
| <b>Used an incorrect public key but recovered quickly</b> | P1, P8, P9, P11, P14 |
| <b>Not applicable</b>                                     | P7, P10              |

**Table 6-15: Participant success at using the correct public keys to envelope lock**

This is the result that most reflects whether the yin-yang key pair metaphor and its presentation were successful in getting users to understand the complementary nature of key pairs, and to avoid the problems with repeated use of the wrong public key that caused so much trouble in the PGP case study. Since all but two participants either never encountered the problem or were able to quickly correct it after the most minimal of feedback, and at least one of the remaining two participants appears to have been derailed by an unrelated usability problem, this is a very good result.

### Other results for use of content locks and signature locks

Table 6-16 shows the results for participant success at consistent correct use of a content lock when sending out the secret memo. Seven of the participants always used a correctly applied content lock. Participants P6 and P10 initially did not use a content lock, but began using content locks correctly once they received a complaint from a team member saying "how do I know that message really came from you?"

|  | <b>Participants</b>           |
|--|-------------------------------|
| <b>Always used correct content lock</b>          | P1, P4, P5, P8, P12, P13, P14 |
| <b>Used correct content lock after complaint</b> | P6, P10                       |
| <b>Used faulty content lock</b>                  | P9                            |
| <b>Did not use content lock</b>                  | P11                           |
| <b>Content locked under someone else's name</b>  | P7                            |

**Table 6-16: Participant use of content lock to authenticate secret memo**

As was discussed earlier for Table 6-13, participant P9 used content locks when appropriate, but applied them incorrectly, so that they protected only her name at the end of the message. She did not receive any feedback from the team members on that point, and continued to make that error throughout the test session. P11 did not use a content lock and did not receive a complaint from a team member to that effect, as she was already ignoring a key authentication complaint due to the misunderstanding discussed earlier. P7 actually applied a content lock using a secret key that she had created in Maria's name; the details of how that came to happen will be discussed shortly.

This result suggests that the content lock metaphor, and the directly manipulable object that it provides, are generally quite successful with users. Only P9's faulty content lock use suggests a usability problem that ideally should be corrected, perhaps through a fadable warning the first time a partially content locked message is sent.

Table 6-17 shows participant success at evaluating whether a requisition had been legally signed by the submitter, meaning that it was protected using a signature lock rather than a content lock. Five participants made the correct decision and approved only the signature locked requisition. Four correctly rejected the plaintext requisition, but erroneously approved the requisition that had only a content lock and not a signature lock. The remaining three either were not given the requisition scenario at all, due to running out of time, or did not have the correct public keys, due to the sequencing problem described earlier.

|   | <b>Participants</b>   |
|---|-----------------------|
| <b>Approved signature locked requisition only</b> | P1, P4, P11, P12, P14 |
| <b>Rejected plaintext requisition only</b>        | P5, P6, P9, P13       |
| <b>Not applicable</b>                             | P7, P8, P10           |

**Table 6-17: Participant success at evaluating signature locks**

This is a somewhat disappointing result, since close to half the participants for this scenario failed to distinguish the content lock from the signature lock. However, most participants were noticeably weary and frustrated by the time they reached this final test scenario, and we suspect that results for this task would have been better if it had been encountered earlier in the test session.

|   | <b>Participants</b> |
|---|---------------------|
| <b>Tried to get “hidden” message out of content lock</b>            | P7                  |
| <b>Applied a content lock that did not cover the important data</b> | P1, P4, P9, P13     |

**Table 6-18: Significant content lock related errors by participants**

Table 6-18 shows results for two participant errors that show some problems with the usability of the content and signature lock metaphor as currently implemented. It is probable that the first one, made by P7, relates to previous exposure to the image of a lock as signaling encryption, or to security in general as being about information hiding. It could probably be ameliorated by adding a corrective message to the verification dialog that appears when a content lock is clicked on. The second error, which we also saw earlier for P9 on the task of sending out the secret memo, appears to have two components: first, that it is not sufficiently clear that a content lock only protects the information that is within its borders, and second, that it is not sufficiently clear how to add one’s own content lock to an already content locked text. Both of those problems should be addressable through fadeable warnings and other minor scaffolding.

### Metaphor related scores on debriefing questionnaire

Table 6-19 shows the metaphor relevant results from the debriefing questionnaire, grouped by concept. For basic key trading, it is interesting that the participants who scored badly on the questionnaire were often those who performed well on the corresponding scenario tasks, and vice versa.

| <b>Concept</b>                  | <b>Score</b> | <b>Participants</b>                   |
|---------------------------------|--------------|---------------------------------------|
| Basic key trading (3 questions) | 3/3          | P5, P10, P12, P13                     |
|                                 | 2/3          | P4, P6, P7, P8, P9, P11, P14          |
|                                 | 1/3          | -                                     |
|                                 | 0/3          | P1                                    |
| Envelope locks (4 questions)    | 4/4          | P5, P6, P12                           |
|                                 | 3/4          | P4, P7, P8, P10                       |
|                                 | 2/4          | P1, P9, P13, P14                      |
|                                 | 1/4          | P11                                   |
|                                 | 0/4          | -                                     |
| Content locks (3 questions)     | 3/3          | P1, P5, P9, P13                       |
|                                 | 2/3          | P4, P6, P7, P8, P10, P12, P14         |
|                                 | 1/3          | P11                                   |
|                                 | 0/3          | -                                     |
| Signature locks (1 question)    | 1/1          | P6, P10, P11                          |
|                                 | 0/1          | P1, P4, P5, P7, P8, P9, P12, P13, P14 |

**Table 6-19: Participant scores on metaphor related questions in debriefing test**

Four participants agreed with the statement “You should give your secret key to the people you want to exchange secure email with,” and five agreed that “You should give your public key only to people you know you can trust,” but they were almost all successful at trading keys during the test. P10, who did not appear to understand key trading at all during the test session, scored perfectly here, while P1, who was one of the most successful participants, scored zero out of three.

As for envelope locks, all participants agreed that “If a message has an envelope lock on it, no-one can read it unless they have the right key.” Five, however, disagreed that “Envelope locks don’t tell you anything about who applied the lock,” and four disagreed that “If you use my public key to put an envelope lock on a message, then no-one can read that message unless they have my matching secret key.” Only half the participants disagreed with the statement, “If I want to put an envelope lock on a message so that only you can read it, I should use my secret key.”

Content locks fared better. All participants correctly disagreed with the statement “Content locks don’t tell you anything about whether a message has been modified since the lock was applied,” and all but two correctly disagreed that “Content locks don’t tell you anything about who applied the lock.” Only five, however, correctly agreed that “Content locks that you make are only useful to people who have your public key.”

Only one of the statements directly addressed signature locks, and nine out of the twelve participants erroneously agreed with it: “If you want to put your legal signature on a secure message, you need to use both a content lock and a signature lock.”

## Significant usability failures

### P7 fatal key import misunderstanding

P7 actually did quite well in the first test scenario, and appeared to be developing a good understanding of key pairs, envelope locks, and content locks. Her fatal problem arose when she began the second test scenario by attempting the task of importing Maria’s public key from the floppy disk. She mistook the “Generate new key pair” button in the Key Pair Vault as being the mechanism for importing a key from the disk, which led her to generate a new key pair under Maria’s name and email address, while maintaining the mistaken belief that she had imported Maria’s public key from the floppy disk. She was never able to correct that misunderstanding, and her possession of a secret key in Maria’s name then contradicted the logic of the system for her and appeared to be the direct cause of her further problems, including her failure to then go on and get public keys from the other team members.

The fatal error here was due to a serious usability flaw; however, this flaw did not relate directly to either the metaphor set or to the staging, but only to the presentation of basic key import and generate functions. It should be correctable by adding stronger reinforcing messages to the dialog boxes encountered for each of those functions.

## P10 general misunderstanding

If there was a test participant who was representative of those people that the Lime user interface does not help, it was P10, who, among other problems, did not ever appear to understand that he needed to get the campaign team members' public keys in the second test scenario. Factors that appeared to contribute to his struggles are as follows:

- This participant actually had some prior exposure to public key based electronic mail security, as he worked in an office that made some use of the security functions in Microsoft Outlook. This appears to have predisposed him to think that one encrypts by attaching a key to the message.
- Possibly also due to his prior experiences with encryption software, when he received feedback that the recipient could not open his envelope locked message, he simply repeated the locking and sending process over and over again, once commenting that this kind of problem happened sometimes in his office too. He never explored the information available to him or appeared to try to correct his understanding of the system.

P10, then, might be a datapoint in support of the argument that the best security system is an invisible one. On the other hand, had P10's previous exposure to security software not led him to expect to proceed with limited understanding, he might have had more success here.

## 6.4 Evaluation summary

The results of the three user tests discussed in this chapter support the following conclusions:

### Staged key certification

The results of the paper-based comparison test and the proof of concept software test both support the usefulness of staging in making key certification manageable for the majority of electronic mail users. In the comparison test, 45% of those who received the staged variant correctly described key certification, versus 10% for the next most successful variant. In the proof of concept test, a solid majority of participants used key certification successfully, and most of those who did not failed because of earlier difficulties not directly related to key certification. Also, those who engaged with the staged aspects of the user interface tended to do well at using key certification.



## Lime visual metaphor set

The paper-based comparison test for metaphors was judged to be a failure of test design, without useful results except perhaps for some degree of validation of the signature lock concept. The evaluation of the Lime visual metaphor set thus rests on the results of the proof of concept test, which appear to be very good. With only two exceptions, one of which appeared to be due to a separate usability failure, participants had little difficulty trading public keys and choosing the correct public keys to create envelope locks with. Nearly all were also able to use content locks correctly, and results from the debriefing questionnaire indicate a high level of understanding of the purpose of content locks. The results for use of signature locks were less promising; however, this may be because those results reflect participant behavior in the final minutes of a lengthy and frustrating test session.

## Usability of visible public key cryptography for electronic mail

The results of the proof of concept test strongly support the argument that visible public key cryptography, if presented using appropriate usability design for security, can be a useful and manageable tool for nearly all electronic mail users. The test conducted here was designed to be challenging: participants were not given a manual, nor were they able to get any help or advice from their correspondents beyond the minimum feedback necessary to tell them that a message they had sent was not authenticated or could not be opened. Those same correspondents set bad examples as well as good ones, sending unauthenticated keys and accepting unauthenticated messages. Despite all this, nearly all the participants were able to use the cryptographic functions appropriately and succeed at the major tasks. It is reasonable to expect that, with the correction of design flaws such as the one that caused trouble for P7, and the addition of good warning messages and other forms of scaffolding, an even higher level of usability may be achieved.

## **CHAPTER 7 Conclusions and future work**

We briefly recap the contributions of this thesis, then outline a variety of interesting approaches for future work.

### **7.1 Contributions**

In this dissertation, we have presented an analysis of the particular usability requirements of computer security, and have shown how those requirements differ significantly from those of general end-user software. Using that analysis, we have derived and presented a number of design principles and two specialized design techniques for creating usable security, and have demonstrated the use of those principles and techniques in the creation of a new user interface for public key cryptography based electronic mail security, called Lime. We have demonstrated through formal user testing that Lime meets a usability threshold sufficient to make sophisticated and flexible security mechanisms manageable by ordinary computer users.

A detailed restatement of the contributions of this thesis is as follows:

Concepts and techniques:

- An analysis of the particular usability requirements of computer security.
- A set of usability design principles particular to computer security.
- A technique called safe staging that allows the user to safely postpone learning how to use a security mechanism.
- A technique for designing effective visual representations of security mechanisms.

Artifacts:

- Lime, which is a working implementation of a user interface design for usable public key cryptography-based secure electronic mail software, coded in Visual C++ and executable in most versions of Microsoft Windows.
- Limeserver, a combination mail and key server simulator, coded in C++, executable in UNIX and intended for use in conducting user tests with Lime.
- A hierarchical design for the visual representation of public key cryptography operations, which may be further specialized and extended for use in a wide variety of security applications.

Experimental results:

- A case study, with formal user testing, of the usability of a commercial security software product which claimed to be highly usable, showing that despite its conventionally friendly graphical user interface, most test participants were unable to use it successfully for basic security tasks.
- A paper-based user test of the effectiveness of presenting public key cryptography using safe staging for the mechanism of key certification, compared to that of two variant presentations that did not use staging, showing that participants who were given the staged variant performed significantly better at describing how key certification would be used.
- An extensive formal user test of Lime, showing that almost all participants were able to successfully use it for most security tasks, and that most participants were able to successfully use it for advanced security tasks.

## 7.2 Future work

This thesis presents a set of design principles and techniques that are applicable to the general problem of creating usable security and thus provides a foundation for a wide variety of future work. We will briefly discuss some of the more interesting research directions that might next be pursued.

### Additional work with Lime

The software implementation of Lime is a valuable tool that could be exploited to get a wide variety of further user test results. Some tests that would be interesting to perform include:

- Additional testing with Lime in its present form, designed for a more focused and thorough evaluation of user understanding of certification locks. For example, we might design a scenario to include an explicit test of whether users understand

that two keys which certify each other are not necessarily any better authenticated than a key with no certification at all.

- Additional testing with slightly modified versions of Lime, to see if identified usability pitfalls can be easily fixed, or to compare various approaches.
- As a field test, Lime could be turned into a fully functional software program and released as shareware, in order to evaluate whether it appeals to the general population.

## Extension to other security applications

Moving beyond public key cryptography for electronic mail security, future research might look to other security applications. Since our tailored metaphor set for public key cryptography was purposely designed as an application-neutral tool, one interesting approach would be to explore further extensions of the lock metaphor to depict other security states, such as access control settings in filesystem security, or the creation of accounts in a networked system.

Another potential ready-made project would be to flesh out, implement and test the staged design for applet security that was discussed in Chapter 4. Staging and metaphor tailoring could also be applied to a wide variety of other security applications.

## Development of additional design techniques

The design principles and techniques that we have presented are not intended to be all that is needed to bridge the gap between general usability design and the usability design needs of computer security. It is likely that there are additional security-specific principles and techniques that need to be developed in order to create a more complete methodology. We think it is likely that some of those techniques will need to address at least the following:

- Guidelines for creating good warning messages for computer security.
- Designing good scaffolding and context help for computer security.
- Moving beyond end-user security to consider how the principles and techniques we have presented are best used or modified for business or other types of organizational security.
- Moving beyond security for PC applications to consider how the principles and techniques we have presented might be translated for user interface design in more constrained environments, such as cell phone displays.

## 7.3 Concluding remarks

Computer security presents a particularly difficult challenge to human computer interaction because many of its usability requirements are fundamentally different from those of other types of end-user software. The failure of naïve user interface designs for security, such as we saw in PGP 5.0, has contributed to widespread pessimism about whether security can ever be made accessible and manageable for most people. Based on the research presented in this thesis, we consider that pessimism to be premature.

If it is possible to make security usable, then it is important to do so. [Norman94] directed our attention to the importance of designing tools that make their users smart rather than stupid. We agree, and further argue that tools should be designed to make their users wise rather than naïve, and empowered rather than dependent.

## APPENDIX A PGP case study materials and data

### A.1 Description of test participants

#### A.1.1 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](mailto: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.1.2 Participant demographics

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

|  |  |
|--|--|
| Gender                                 | 6 female<br>6 male   |
| Age                                    | 3 age 25 or younger<br>3 age 26 to 35<br>3 age 36 to 45<br>3 age 45 or older   |
| Highest education level <sup>14</sup>  | 2 had some college<br>4 had undergraduate degrees<br>4 had some graduate school<br>2 had graduate degrees  |
| Education or career area <sup>15</sup> | 2 did computer programming<br>4 did biological science (pharmacy, biology, medicine)<br>4 did humanities or social science (education, business, sociology, English)<br>2 did fine arts (graphic design) |

---

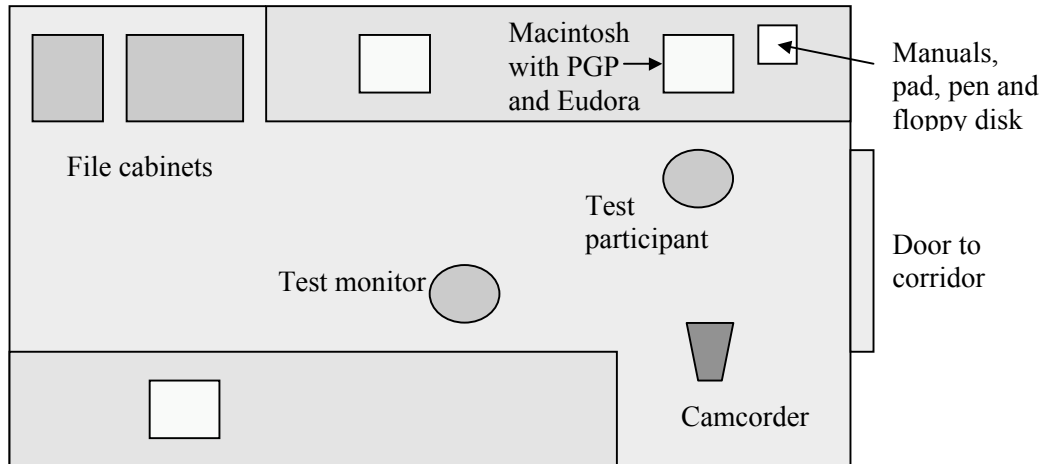
<sup>14</sup> The original test plan called for some participants with only a high school education, but none responded to the advertisements.

<sup>15</sup> 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.

## A.2 Description of testing process

### A.2.1 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.



## A.2.2 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, and asked to read it and sign one copy.
- 2) The written briefing was read aloud to the participant, 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;
  - 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, 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.

## A.2.3 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:

- Generate a key pair of their own.
- 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.
- 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.
- 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.

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

- 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.
- 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.
- 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.

## A.2.4 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*. The test monitor then thanked the participant for their help, paid them the \$20 in cash, and ended the test session.

## A.3 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 | Fetches 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.
- When preparing to send his key to the key server, he expressed worry that he couldn't tell whether the key in the PGPKKeys 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 PGPKKeys.

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 | Fetches 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 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 PGPKKeys 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 | Fetches 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 | Fetches 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 PGPKKeys, 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 | Fetches 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 | Fetches 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.

## A.4 Test materials

### A.4.1 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.



## A.4.2 Initial task description

The campaign manager is Maria Page, [mpage@wanton.trust.cs.cmu.edu](mailto:mpage@wanton.trust.cs.cmu.edu).

The other members of the campaign team are:

Paul Butler, [butler@wanton.trust.cs.cmu.edu](mailto:butler@wanton.trust.cs.cmu.edu)  
Ben Donnelly, [bend@wanton.trust.cs.cmu.edu](mailto:bend@wanton.trust.cs.cmu.edu)  
Sarah Carson, [carson@wanton.trust.cs.cmu.edu](mailto:carson@wanton.trust.cs.cmu.edu)  
Dana McIntyre, [dmi@wanton.trust.cs.cmu.edu](mailto: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<sup>16</sup>. Don't forget to "think aloud" as much as you can.

---

<sup>16</sup> 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.

### A.4.3 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?

## **APPENDIX B Staging comparison test materials and data**

This appendix contains test materials and collated data from the comparison of my staged user interface design against two alternatives, as described in 6.1. The contents are as follows:

Questions used to screen recruited participants.

Participant initial briefing.

Presentation for staged (Lime) variant

Presentation for PGP variant

Presentation for hidden key pair (SSL) variant

Questions given to participants, with collated responses, including debriefing questionnaire.

## B.1 Participant screening questions

Your age:

Your education level (HS diploma, some college, BS/BA, MS/MA, PhD):

Your major or profession or main field of expertise:

How long you've been using email:

The name of any computer security software you've used in the past:

Have you ever studied number theory or cryptography?

Are you familiar with the difference between public and private key cryptography? If yes, please describe briefly.

## B.2 Briefing given to participants

The purpose of this test is to compare different ways of explaining a computer security system to see which way people understand it best.

The test will have three parts. First, I will give you a two page description of a software program for computer security, and ask you to read over the description until you feel like you understand how it works, taking as much time as you need.

Second, I will give you, one at a time, five sets of questions about how to use the software, and ask you to answer them in writing, again taking as much time as you need. You will still be able to look at the written description while you are answering the questions. Please assume, when answering the questions, that individual computers are reasonably secure, but that the network is not: there are no secure servers or secure intranets available, so all the security must come from the program you have been given.

Third, I will give you a short questionnaire that asks you about your reactions to the description.

While we are doing the test, remember that you are helping us test this design – you yourself are not being tested, so please do not worry about whether you are doing well. Also, please do not feel any pressure to answer as if you like or dislike the design you are shown – what is important to us is to get accurate information, not to prove that a particular design is good or bad.

*I will now give you the description of the software; please read it over and let me know when you are ready for the first question.*

## B.3 Presentation variants

This section reproduces the three variant presentations given to the user. The first is the staged variant that corresponds to Lime, the second is the unstaged variant that corresponds to PGP, and the third is the unstaged variant that conceals the key pair, corresponding roughly to SSL.

## YOUR ELECTRONIC MAIL SECURITY SOFTWARE

### Security functions

Your electronic mail security software provides functions for protecting your mail messages against unauthorized reading (eavesdropping) and unauthorized modification (tampering or forgery).

To protect a message against unauthorized reading, use the **make-unreadable** function on it. Then an authorized person will need to use the matching **make-readable** function in order to read the message, and no-one else will be able to read it at all.

To protect a message against unauthorized modification, including forgery, use the **make-tamperproof** function on it. People who view the message will then be able to use the matching **check-tamperproofing** function to see who tamperproofed the message and to verify that no later modification has occurred.

Each of these four functions must be used with a **security token**.

### Security tokens

Each person who uses the security software must have their own matched pair of security tokens, consisting of one **secret token** and one **public token**. Secret tokens must be carefully protected so that only their owners have access to them. Public tokens, however, are meant to be freely distributed and traded, so you should give your public token to everyone you send messages to and get their public tokens in return. You can use the **generate-tokens** function to make a token pair for yourself.

Secret tokens are used for proving identity. They can be used this way while still remaining secret because of the special power of a matched token pair. If a public token is used to do make-unreadable on a message, then only someone with the matching secret token can make that message readable again. Likewise, if a public token can be used to do check-tamperproofing on a message, then that proves that the tamperproofing was made with the matching secret token.

- To protect a message so that only a specific person can read it, use that person's public token to do make-unreadable on it.
- To read a protected message that has been sent to you, use your secret token to do make-readable on it.
- To protect a message so that people can be sure it hasn't been forged, or changed since you protected it, use your secret token to do make-tamperproof on it.
- To verify that no-one has changed a message that has been sent to you since it was tamperproofed, and to verify the identity of the person who tamperproofed it, use that person's public token to do check-tamperproofing on it.

### **Protecting your secret token**

Because your secret token is your proof of identity, you need to carefully protect it against theft or loss. When you create your token pair, your software will ask you to choose a pass phrase (like a password, but longer, since it has multiple words). Your software will not let anyone access your secret token unless they have that pass phrase, so choose something that you will find easy to remember, but that other people won't be able to guess. Then make a backup copy of your token pair and store it in a safe place.

### **Trading public tokens to get basic security**

The security of your messages depends on having the right public token for each person. If an attacker can trick you into thinking their public token belongs to someone you send messages to, then the attacker can read protected messages you send to that person, and forge tamperproofed messages from that person to you. When you trade public tokens, you need to take precautions to make sure you aren't being tricked.

The simplest way to trade public tokens is usually to send them in mail messages or put them up on personal web pages for downloading. The risk is that an attacker could set up a fake web page or forge an email message so that it appears to be from someone you know. For basic security, protect yourself against these kinds of tricks by asking common sense questions. Have you been to this person's web page before, and is it at a web address you know that person uses? Does the message with the token in it sound like that person, and mention things that person would know? Does it come from an email address that you know that person uses? Likewise, when you send your public token to other people, include a note that will help them be sure the message came from you.

This level of security is enough to protect your messages against random eavesdropping and simple forgery, and against attackers who are looking for general vulnerabilities and have no reason to work hard to target your messages in particular. If your messages contain very sensitive or valuable data, or if you have some other reason to think an attacker might want to single you out as a target, then you should consider a stronger level of security. You may also need to use the stronger level if you do not know the other person well enough for the common sense questions to be useful.

### **Trading public tokens to get stronger security**

The most secure way to trade public tokens is to put them on floppy disks and meet in person to trade them, so that you can be absolutely sure that the public token you get does belong to the person who handed it to you. Once you have at least one public token whose ownership you are absolutely sure of, you can use that to help you get public tokens through a second way that is only slightly less secure.

This second way involves trusting the owner of that public token to tell you who other public tokens belong to, via tamperproofed messages. If you are absolutely sure you

have the right public token for person A, and you trust person A's judgement, then a tamperproofed message from person A stating that person A has made absolutely sure that the included public token belongs to person B may be considered almost as secure as meeting with person B yourself. And as long as the message containing person B's token and statement from person A is tamperproofed by person A, it can be distributed through any web page or public database without worrying about further security.



## YOUR ELECTRONIC MAIL SECURITY SOFTWARE

### Security functions

Your electronic mail security software provides functions for protecting your mail messages against unauthorized reading (eavesdropping) and unauthorized modification (tampering or forgery).

To protect a message against unauthorized reading, use the **make-unreadable** function on it. Then an authorized person will need to use the matching **make-readable** function in order to read the message, and no-one else will be able to read it at all.

To protect a message against unauthorized modification, including forgery, use the **make-tamperproof** function on it. People who view the message will then be able to use the matching **check-tamperproofing** function to see who tamperproofed the message and to verify that no later modification has occurred.

Each of these four functions must be used with a **security token**.

### Security tokens

Each person who uses the security software must have their own matched pair of security tokens, consisting of one **secret token** and one **public token**. Secret tokens must be carefully protected so that only their owners have access to them. Public tokens, however, are meant to be freely distributed and traded, so you should give your public token to everyone you send messages to and get their public tokens in return. You can use the **generate-tokens** function to make a token pair for yourself.

Secret tokens are used for proving identity. They can be used this way while still remaining secret because of the special power of a matched token pair. If a public token is used to do make-unreadable on a message, then only someone with the matching secret token can make that message readable again. Likewise, if a public token can be used to do check-tamperproofing on a message, then that proves that the tamperproofing was made with the matching secret token.

- To protect a message so that only a specific person can read it, use that person's public token to do make-unreadable on it.
- To read a protected message that has been sent to you, use your secret token to do make-readable on it.
- To protect a message so that people can be sure it hasn't been forged, or changed since you protected it, use your secret token to do make-tamperproof on it.
- To verify that no-one has changed a message that has been sent to you since it was tamperproofed, and to verify the identity of the person who tamperproofed it, use that person's public token to do check-tamperproofing on it.

### **Protecting your secret token**

Because your secret token is your proof of identity, you need to carefully protect it against theft or loss. When you create your token pair, your software will ask you to choose a pass phrase (like a password, but longer, since it has multiple words). Your software will not let anyone access your secret token unless they have that pass phrase, so choose something that you will find easy to remember, but that other people won't be able to guess. Then make a backup copy of your token pair and store it in a safe place.

### **Trading public tokens**

The security of your messages depends on having the right public token for each person. If an attacker can trick you into thinking their public token belongs to someone you send messages to, then the attacker can read protected messages you send to that person, and forge tamperproofed messages from that person to you. When you trade public tokens, you need to take precautions to make sure you aren't being tricked.

The most secure way to trade public tokens is to put them on floppy disks and meet in person to trade them, so that you can be absolutely sure that the public token you get does belong to the person who handed it to you. Once you have at least one public token whose ownership you are absolutely sure of, you can use that to help you get public tokens through a second way that is only slightly less secure.

This second way involves trusting the owner of that public token to tell you who other public tokens belong to, via tamperproofed messages. If you are absolutely sure you have the right public token for person A, and you trust person A's judgement, then a tamperproofed message from person A stating that person A has made absolutely sure that the included public token belongs to person B may be considered almost as secure as meeting with person B yourself. And as long as the message containing person B's token and statement from person A is tamperproofed by person A, it can be distributed through any web page or public database without worrying about further security.

## YOUR ELECTRONIC MAIL SECURITY SOFTWARE

### Security functions

Your electronic mail security software provides functions for protecting your mail messages against unauthorized reading (eavesdropping) and unauthorized modification (tampering or forgery).

To protect a message against unauthorized reading, use the **make-unreadable** function on it. Then an authorized person will need to use the matching **make-readable** function in order to read the message, and no-one else will be able to read it at all.

To protect a message against unauthorized modification, including forgery, use the **make-tamperproof** function on it. People who view the message will then be able to use the matching **check-tamperproofing** function to see who tamperproofed the message and to verify that no later modification has occurred.

Each of these four functions must be used with a **security token**.

### Security tokens

Each person who uses the security software must have a security token, which can be freely distributed and traded. You should give your token to everyone you send messages to and get their tokens in return. You can use the **generate-token** function to make a security token for yourself.

- To protect a message so that only a specific person can read it, use that person's token to do make-unreadable on it.
- To read a protected message that has been sent to you, use your token to do make-readable on it.
- To protect a message so that people can be sure it hasn't been forged, or changed since you protected it, use your token to do make-tamperproof on it.
- To verify that no-one has changed a message that has been sent to you since it was tamperproofed, and to verify the identity of the person who tamperproofed it, use that person's token to do check-tamperproofing on it.

### Protecting your security token

You can trade tokens freely, but in order to use a token to do make-readable or make-tamperproof, you have to prove to the software that it belongs to you. When you create your token, your software will ask you to choose a pass phrase (like a password, but longer, since it has multiple words). No one can use your token to do make-readable or make-tamperproof unless they know that pass phrase, so choose something that you will find easy to remember, but that other people won't be able to guess. Then make a backup copy of your token and store it in a safe place.

## **Trading security tokens**

The security of your messages depends on having the right token for each person. If an attacker can trick you into thinking their token belongs to someone you send messages to, then the attacker can read protected messages you send to that person, and forge tamperproofed messages from that person to you. When you trade tokens, you need to take precautions to make sure you aren't being tricked.

The most secure way to trade tokens is to put them on floppy disks and meet in person to trade them, so that you can be absolutely sure that the token you get does belong to the person who handed it to you. Once you have at least one token whose ownership you are absolutely sure of, you can use that to help you get tokens through a second way that is only slightly less secure.

This second way involves trusting the owner of that token to tell you who other tokens belong to, via tamperproofed messages. If you are absolutely sure you have the right token for person A, and you trust person A's judgement, then a tamperproofed message from person A stating that person A has made absolutely sure that the included token belongs to person B may be considered almost as secure as meeting with person B yourself. And as long as the message containing person B's token and statement from person A is tamperproofed by person A, it can be distributed through any web page or public database without worrying about further security.

## B.4 Questions and collated results

### Participant demographics

#### Hidden key pair (SSL) variant

| P#  | age | education    | expertise                                   |
|-----|-----|--------------|---|
| P9  | 20  | in college   | Information Systems                         |
| P35 | 28  | M.S. degree  | Electrical Engineering                      |
| P6  | 20  | in college   | Electrical and Computer Engineering         |
| P34 | 22  | some college | Mathematics                                 |
| P18 | 45  | M.S. degree  | Industrial Engineering, Information Science |
| P41 | 19  | some college | Chemistry/Biology                           |
| P12 | 27  | B.S. degree  | technical writing                           |
| P27 | 21  | some college | chemical engineering                        |
| P15 | 20  | some college | public policy and management                |

#### PGP variant

| P#  | age | education    | expertise                           |
|-----|-----|--------------|-------------------------------------|
| P29 | 20  | some college | Mechanical Engineering              |
| P32 | 19  | some college | architecture                        |
| P36 | 20  | some college | computer science                    |
| P38 | 19  | some college | electrical and computer engineering |
| P31 | 24  | B.A.         | architecture                        |
| P17 | 21  | some college | business administration             |
| P8  | 19  | some college | business                            |
| P30 | 27  | B.S. degree  | Physics/Chemistry/Languages         |
| P11 | 18  | some college | Psychology and English              |
| P13 | 24  | M.S. degree  | Chemical engineering                |

#### Staged (Lime) variant

| P#  | age | education            | expertise                    |
|-----|-----|----------------------|------------------------------|
| P10 | 30  | M.S. degree          | Public Policy and Management |
| P4  | 20  | some college         | computer science             |
| P5  | 19  | some college         | computer science             |
| P42 | 19  | some college         | Mechanical Engineering       |
| P33 | 24  | some graduate school | Physical Biochemistry        |
| P37 | 18  | some college         | computer science             |
| P16 | 19  | some college         | Bio/Psych                    |
| P14 | 41  | M.A. degree          | Administration               |

|     |    |              |                             |
|-----|----|--------------|-----------------------------|
| P7  | 18 | some college | Chemical Engineering        |
| P40 | 23 | B.S. degree  | Neuroscience and Psychology |
| P28 | 30 | M.A. degree  | History                     |

**Time spent on each portion of test session (minutes), not including debriefing questionnaire**

**Hidden key pair (SSL) variant**

| P#  | reading prior to requesting first question | Question 1 | Question 2 | Question 3 | Question 4 | Question 5 | total |
|-----|--|------------|------------|------------|------------|------------|-------|
| P9  | 4  | 5          | 8          | 3          | 2          | 4          | 26    |
| P35 | 6  | 4          | 10         | 9          | 6          | 6          | 41    |
| P6  | 6  | 4          | 12         | 7          | 5          | 9          | 43    |
| P34 | 7  | 4          | 6          | 15         | 3          | 7          | 42    |
| P18 | 7  | 1          | 10         | 5          | 5          | 5          | 33    |
| P41 | 4  | 1          | 7          | 6          | 6          | 7          | 31    |
| P12 | 4  | 1          | 1          | 6          | 3          | 11         | 26    |
| P27 | 2  | 2          | 4          | 3          | 2          | 2          | 15    |
| P15 | 3  | 4          | 3          | 1          | 2          | 2          | 15    |

**PGP variant**

| P#  | reading presentation prior to requesting first question | Question 1 | Question 2 | Question 3 | Question 4 | Question 5 | total |
|-----|---|------------|------------|------------|------------|------------|-------|
| P29 | 4   | 2          | 8          | 2          | 3          | 3          | 22    |
| P32 | 9   | 4          | 6          | 4          | 8          | 10         | 41    |
| P36 | 5   | 4          | 2          | 3          | 3          | 3          | 20    |
| P38 | 7   | 8          | 3          | 3          | 2          | 3          | 26    |
| P31 | 9   | 2          | 9          | 13         | 4          | 7          | 44    |
| P17 | 4   | 2          | 1          | 5          | 6          | 4          | 22    |
| P8  | 4   | 2          | 3          | 2          | 3          | 3          | 17    |
| P30 | 4   | <1         | 10         | 7          | 7          | 3          | 31    |
| P11 | 5   | 2          | 8          | 4          | 5          | 5          | 29    |
| P13 | 8   | 2          | 8          | 6          | 8          | 5          | 37    |

**Staged (Lime) variant**

| P# | reading presentation prior to requesting first question | Question 1 | Question 2 | Question 3 | Question 4 | Question 5 | total |
|----|---|------------|------------|------------|------------|------------|-------|
|----|---|------------|------------|------------|------------|------------|-------|

|     |    |   |    |    |   |   |    |
|-----|----|---|----|----|---|---|----|
| P10 | 11 | 9 | 17 | 12 | 9 | 3 | 71 |
| P4  | 7  | 3 | 2  | 3  | 4 | 5 | 24 |
| P5  | 6  | 3 | 8  | 8  | 6 | 8 | 39 |
| P42 | 3  | 1 | 4  | 3  | 1 | 3 | 15 |
| P33 | 4  | 1 | 5  | 6  | 4 | 6 | 26 |
| P37 | 5  | 3 | 5  | 2  | 6 | 7 | 28 |
| P16 | 5  | 1 | 6  | 5  | 6 | 6 | 29 |
| P14 | 5  | 1 | 7  | 6  | 6 | 9 | 34 |
| P7  | 6  | 8 | 7  | 8  | 4 | 7 | 40 |
| P40 | 7  | 1 | 6  | 4  | 3 | 4 | 27 |
| P28 | 4  | 1 | 1  | 4  | 2 | 3 | 15 |

### Question 1

You need to send an email message to your housemates reminding them to buy lightbulbs, since your household is nearly out.

(1) Would you, in real life, think it was worth putting in some extra time to make this message secure, rather than simply sending it in a regular email? If yes, how much extra time (in seconds, minutes, hours, or days) would you think it was worth?

#### Hidden key pair (SSL) variant

|     |  |
|-----|--|
| P9  | It isn't worth making the message secure because you just need extra lightbulbs. It's easier to just send a regular email to the other housemates. |
| P35 | It's not worthy to waste time on it.   |
| P6  | No   |
| P34 | Yes, it would be worth 1 minute.   |
| P18 | No.  |
| P41 | No... it's lightbulbs... if someone else wants to hear about our plight, let them.   |
| P12 | no, not worth it   |
| P27 | I wouldn't worry about making this email secure. The information being sent is not important to worthy security.                                   |
| P15 | It depends who it was going to the content of the email. 10 seconds.   |

#### PGP variant

|     |  |
|-----|--|
| P29 | in real life, it is not worth putting extra time to make the message secure. lightbulbs are not that important and you can always make a quick trip to the store if they forget. |
| P32 | no, it's not anything private or personal.   |
| P36 | Personally, I would send this kind of message w/regular email. Since I will not be specifically mentioning the address of my house, I don't think                                |

|     |   |
|-----|---|
|     | a simple "buy me a light bulb" email needs security.              |
| P38 | No  |
| P31 | It is not worth for this particular occasion (to buy lightbulbs)  |
| P17 | No  |
| P8  | I do not think it needs to be made secure. I'd use regular email. |
| P30 | No.   |
| P11 | No, I do not find it especially necessary.                        |
| P13 | No  |

### Staged (Lime) variant

|     |  |
|-----|--|
| P10 | No, I don't think it was worth putting in some extra time for this kind of message.  |
| P4  | Not worth putting extra effort to make it secure.  |
| P5  | No: Not worth it.  |
| P42 | No   |
| P33 | No.  |
| P37 | I would not be willing to spend extra time to secure an email about buying lightbulbs.   |
| P16 | No, this is not a personal email.  |
| P14 | No, not really.  |
| P7  | Making a message about buying lightbulbs secure wouldn't be a priority for me. If making it secure took seconds, however, then I would probably do it. |
| P40 | no I would not think it was worth making it secure.  |
| P28 | No   |

(2) If you answered “yes” to question 1, then can you tell, from the software description you were given, which tokens and which functions you and your housemates would each need to use? If yes, please list them.

### Hidden key pair (SSL) variant

|     |  |
|-----|--|
| P9  | <blank>  |
| P35 | <blank>  |
| P6  | <blank>  |
| P34 | I would need to use his/her token w/make-unreadable and my token w/make-tamperproof. |
| P18 | <blank>  |
| P41 | <blank>  |
| P12 | <blank>  |
| P27 | X  |
| P15 | make-unreadable, make-readable, make-tamperproof, check-tamperproof                  |



### PGP variant

|     |         |
|-----|---------|
| P29 | N/A     |
| P32 | <blank> |
| P36 | <blank> |
| P38 | <blank> |
| P31 | <blank> |
| P17 | <blank> |
| P8  | <blank> |
| P30 | <blank> |
| P11 | N/A     |
| P13 | <blank> |

### Staged (Lime) variant

|     |   |
|-----|---|
| P10 | <blank>   |
| P4  | Use my housemate's public token & make-unreadable function.   |
| P5  | <blank>   |
| P42 | <blank>   |
| P33 | <blank>   |
| P37 | <blank>   |
| P16 | <blank>   |
| P14 | <blank>   |
| P7  | I would use my housemates public token to "make unreadable" (making it tamper-proof wouldn't seem worth the effort). then my housemates would use their secret tokens to "make readable". |
| P40 | <blank>   |
| P28 | <blank>   |

(3) If you answered “yes” to question 1, then can you tell, from the software description you were given, what steps you and your housemates would each need to take to get those tokens? If yes, please list them.

### Hidden key pair (SSL) variant

|     |  |
|-----|--|
| P9  | You would have to make the token, and then pass it along to the others.<br>But if you have to go see them just to pass along a token, then why not just tell them when you see them? |
| P35 | <blank>  |
| P6  | <blank>  |
| P34 | We would have to have swapped tokens back at the house on floppy.  |
| P18 | <blank>  |
| P41 | <blank>  |

|     |   |
|-----|---|
| P12 | <blank>                                 |
| P27 | X                                       |
| P15 | make a pass phrase (or a long password) |

### PGP variant

|     |         |
|-----|---------|
| P29 | N/A     |
| P32 | <blank> |
| P36 | <blank> |
| P38 | <blank> |
| P31 | <blank> |
| P17 | <blank> |
| P8  | <blank> |
| P30 | <blank> |
| P11 | N/A     |
| P13 | <blank> |

### Staged (Lime) variant

|     |   |
|-----|---|
| P10 | <blank>   |
| P4  | I can d/L his public token on the web & send email to him w/that token & then make-unreadable function.   |
| P5  | <blank>   |
| P42 | <blank>   |
| P33 | <blank>   |
| P37 | <blank>   |
| P16 | <blank>   |
| P14 | <blank>   |
| P7  | We would use the "generate-tokens" function to first make the token pair. Then, since we live together, we would exchange public tokens on floppy disks to ensure maximum token security. |
| P40 | <blank>   |
| P28 | <blank>   |

(4) Are there any comments you would like to make?

### Hidden key pair (SSL) variant

|     |   |
|-----|---|
| P9  | For simple things like passing on a message to get lightbulbs, it isn't necessary to use these tokens. Maybe in top secret business deals, but even in those, it's always best to meet in private to assure that it's confidential. |
| P35 | Maybe when you create a new mail you can have it set to make it secure by default for anyone (or selected persons) in your personal mailing list.   |
| P6  | No  |

|     |                  |
|-----|------------------|
| P34 | No               |
| P18 | Sort of trivial. |
| P41 | <blank>          |
| P12 | <blank>          |
| P27 | <blank>          |
| P15 | Nope             |

### PGP variant

|     |  |
|-----|--|
| P29 | There are something that should be secured, but other simple messages are fine.  |
| P32 | <blank>  |
| P36 | In general, I would say less than 10% of my outgoing email so far needed some kind of protection (over 90% I was comfortable sending w/out using any security program) |
| P38 | The secure process is really long and annoying for just this simple notes.   |
| P31 | <blank>  |
| P17 | It doesn't seem worthwhile to make the message secure if its merely about lightbulbs.  |
| P8  | No :)  |
| P30 | <blank>  |
| P11 | This software is too complicated for simple, day-to-day communications such as the task/errand stated above.   |
| P13 | No   |

### Staged (Lime) variant

|     |  |
|-----|--|
| P10 | I think it would be worth securing my message only if when I think the cost of my message being tempered is larger than the value of time I put in it. Since lightbulbs is not worthy very much, if somebody temped the message, and my housemates didn't buy any, I can send him email several days later to ask him to do it. If he buy more than we need, it can be stored as supplies. If he buys less, he can buy some next time. And I don't think such kind of message will become a target of tampering. |
| P4  | no.  |
| P5  | The sentence "If a public token is used to do make-unreadable on a message..." confused me a few times when I read it, in part because you end the line with 'make-' and also the 'do make' is a bit awkward. Other than this the reading was understandable and clearly written.  |
| P42 | <blank>  |
| P33 | <blank>  |
| P37 | I might be interested in secure email for other types of email however.  |
| P16 | <blank>  |
| P14 | <blank>  |
| P7  | Using security software for something like this doesn't seem all that important.   |
| P40 | <blank>  |

## Question 2

You want to send an email message to an old friend from school to bring them up to date on some stressful developments in your personal life. You haven't talked to them in a year or so, since they now live on the other side of the country, but you have their email address at the company they currently work for.

(5) Would you, in real life, think it was worth putting in some extra time to make this message secure, rather than simply sending it in a regular email? If yes, how much extra time (in seconds, minutes, hours, or days) would you think it was worth?

### Hidden key pair (SSL) variant

|     |  |
|-----|--|
| P9  | Yes, depending on how personal these situations were and how it would affect you if the info got out, I would spend enough time to make it secure. If it was very private, I would spend up to days making it secure.                  |
| P35 | Yes, I would take some seconds (5-15) to make sure it is secure.   |
| P6  | yes, 10 sec  |
| P34 | Yes, 10 minutes  |
| P18 | Yes 30 min.  |
| P41 | Maybe... but first I would have to get the tokens to + from them + since they live on the other side of hte country - that would take a while - I could just send them a regular letter - but if they already had the tokens - I would |
| P12 | it would only be worth it if I felt that an unauthorized party could cause me harm & grief by having that information. unless I was some kind of celebrity, it's probably not worth it.  |
| P27 | I would put in a few extra minutes (2-4) to protect the info, since some of it may be of personal manner.  |
| P15 | Yes, 10 seconds.   |

### PGP variant

|     |   |
|-----|---|
| P29 | It may be worth to put in extra time for security. About a couple of minutes, maybe.  |
| P32 | Yes. 30 minutes.  |
| P36 | I really don't worry about my casual personal life being public knowledge so I guess I would send this message w/regular email                |
| P38 | Yes, less than an hour.   |
| P31 | Assuming that after the first effort, the next times will be easier, I would spend the necessary extra time (1-2 hour at most!) to set it up. |
| P17 | No  |
| P8  | Yes, I'd spend an extra 5 minutes   |
| P30 | perhaps, if I had some reason to be concerned about, e.g. my partner, boss or the press having details of my personal life. five minutes      |
| P11 | Yes, I might consider putting in a slight amount of extra time -- i.e. no   |

|     |                           |
|-----|---------------------------|
|     | more than a minute or so. |
| P13 | Yes, 10 minutes           |

### Staged (Lime) variant

|     |  |
|-----|--|
| P10 | I think it is worth putting in some extra time to protect it from eavesdropping. Maybe less than 10 minutes. If more than that, I rather send them an email to ask their tel. no and mail address so as I can talk or write to them other than by email. |
| P4  | no. probably not.  |
| P5  | Yes. If it were the 1st time and I had to set things up I'd spend maybe 5-10 minutes max. As a routine thing to do I'd only spend 10-30 seconds max each time.   |
| P42 | Yes, 30 seconds.   |
| P33 | yes <= 1 min   |
| P37 | I would probably be willing to spend an extra 5 minutes or so to secure the email.   |
| P16 | Yes, I would spend about 2 minutes.  |
| P14 | Yes, several minutes would be acceptable. More time than that, I would just email a simple greeting and request a telephone number or for them to call me at work to exchange home telephone numbers.  |
| P7  | Maybe a few seconds extra time, but I wouldn't go <u>too</u> far out of my way unless I suspected that someone was stalking me or something.   |
| P40 | yes I would. since it is very personal I would take a day to be sure it was secure if I needed to, but not much longer.  |
| P28 | No   |

(6) If you answered “yes” to question 5, then can you tell, from the software description you were given, which tokens and which functions you and your friend would each need to use? If yes, please list them.

### Hidden key pair (SSL) variant

|     |   |
|-----|---|
| P9  | You would have to use the make-unreadable function and the friend make-readable function. To make sure that no-one can tamper with the letter you would use the make-tamperproof function. And the friend, the check-tamperproofing function. Then you would generate a token, make a password, tell the friend the password and then pass the token along. |
| P35 | I will use the make-unreadable function. I would need to have in advance my old friend's token to be able to write him the mail. My friend only needs his token to be able to read the mail.  |
| P6  | <u>Me</u> : <u>My friend's token</u> as input of " <u>make-unreadable</u> " function   <u>My friend</u> : <u>His own token</u> as input of " <u>make-readable</u> " function.   |
| P34 | I would have to use my friend's token w/make-unreadable and my token with make-tamperproof.   |
| P18 | Yes. Need to give that person your token, and to get that person's token from them. Need functions make-unreadable and make-readable.   |
| P41 | - set the makeunreadable function - set the make tamperproof function   |

|     |  |
|-----|--|
| P12 | <blank>  |
| P27 | I would need my friend's token to make-unreadable on the message. The friend would also have my token to make sure it was not tampered with. |
| P15 | make readable, make unreadable   |

### PGP variant

|     |  |
|-----|--|
| P29 | Public tokens. Me: to protect my message. Use public token to make-unreadable. Friend: use his secret token to make the message readable.  |
| P32 | me - my friend's public token, make-unreadable. my friend - secret token, make-readable.   |
| P36 | <blank>  |
| P38 | Yes, I need to use make-unreadable, he need to use make-readable and generate token, he need to have secret token, I use his public token for make unreadable, he use his secret token for make readable.  |
| P31 | I would need his 'public token'. He would need his 'public token'.   |
| P17 | <blank>  |
| P8  | I'd make it make-unreadable by using his/her's public token, and ask them to use their secret token to protect it after they get it.   |
| P30 | <u>I'd need my friend's public token to "make-unreadable". He/she'd need his/her private token to read it.</u> If I also wanted to tamperproof the message, <u>I'd need my private token and the friend would need my public token.</u>                      |
| P11 | I would need to find someone trustworthy to send me one of my friend's public tokens in order for her (friend) to read the mail I sent, we could use the make-unreadable (me)/readable (her) and make-tamperproof (me)/check tamperproofing (her) functions. |
| P13 | I would need to use: my friend's public token to make-unreadable, my secret token to make-tamperproof. He/she would need to use: my public token to check-tamperproof, his/her secret token to make-readable.  |

### Staged (Lime) variant

|     |   |
|-----|---|
| P10 | For me: use my friend's public token to do an make-unreadable function on it. For my friend: He use his secret token to do make-readable function on this email.                        |
| P4  | use his public token and make-unreadable function.  |
| P5  | Yes. I would have to have my friend's public token, and I would need to 'make-unreadable' with that token. My friend would need their private token, and would 'make-readable' with it. |
| P42 | I would have to use the make-unreadable function w/his public token so that he can only read it w/his private token using make-readable function.                                       |
| P33 | yes. I would use my friend's public token to "make unreadable". He would use his own private token to "make readable"   |
| P37 | I would use generate-tokens to get my pair, and I would use the make-unreadable function. My friend would use make-readable to view the email.  |
| P16 | Yes, I would have to use my friend's public token to make it unreadable. <del>would also have to use the tamper</del> I would have to use the make-unreadable function.                 |
| P14 | I would at least generate a token pair in order to <u>make-unreadable</u> , if not also <u>make-tamperproof</u> . Then I would priority-mail or UPS a diskette with                     |

|     |   |
|-----|---|
|     | my public token to my friend at his business address, requesting that he create a token pair, and send me his/her public token back to me on diskette so I could read their responses.  |
| P7  | I would use my friend's public token to 'make-unreadable', and my friend would use his/her secret token to 'make readable'. Making it tamperproof doesn't seem worth it.  |
| P40 | tokens: me - her public, my secret. her - my public, her secret.<br>functions - make unreadable, make readable, make tamperproof, check tamperproofing. I would want us both to have a public + secret token. I would need her public token to make the message unreadable and she would need a secret token. When she responds she would need my public token + I would need a secret token. |
| P28 | <blank>   |

(7) If you answered “yes” to question 5, then can you tell, from the software description you were given, what steps you and your friend would each need to take to get those tokens at an appropriate level of security? If yes, please list them.

#### Hidden key pair (SSL) variant

|     |   |
|-----|---|
| P9  | The best way is to put the on disk and pass it along in person.   |
| P35 | To get my friend's token I would have to either: 1) send an unsecure mail asking my friend to send the token in a floppy disk through certified mail, 2) Ask my friends, for which I already have tokens, if any of them has the token of my old friend so that they will forward it to me in a secured make-unreadable message.  |
| P6  | Me: I can get his token from him personally or by a secure trusted intermediate person.   My friend meet with me or give his token to a secure trusted entity.  |
| P34 | (If I knew some else (B) who knew my friend's token and my friend knew that person's (B's) token) Then I could send them (B) a message securely with my token asking for my friend's token. Otherwise I'd have to meet them (my friend) in person.  |
| P18 | <del>You can't physically exchange tokens. Must use an intermediate source that can send you the other person's token, and you must give your token to an other &lt;---&gt; A &lt;-----&gt; B &lt;---&gt; you</del> Connection must be setup first between A & B. You can give your token to B, B can transmit it to A and A can give it to that other person. And in reverse to setup the scheme.<br><del>This has to</del> The Connection other <-->A A <--> B and B <--> you has to be setup before this can happen. |
| P41 | - use my token w/the "make readable" function - if they want to check for forgery - use my token w/the "check tamperproofing" function  |
| P12 | <blank>   |
| P27 | Maybe a registered letter if you can't meet with that person. But that may take too much time, and if you're sending a letter, you might as well just write the info in the letter and not in the e-mail.   |
| P15 | No.   |

#### PGP variant

|     |  |
|-----|--|
| P29 | Best way would be in person, but because we are far away that's not possible. I would send my friend a tamperproof message stating that I am |
|-----|--|

|     |  |
|-----|--|
|     | sure that included is the public token to my friend. A - me, B - friend.   |
| P32 | Before I send, I should use my friend's public token to make-unreadable. Then after he receives it, he needs to use his secret token to make-readable.   |
| P36 | <blank>  |
| P38 | No   |
| P31 | We would individually <u>put</u> our 'public tokens' on separate floppy disks and <u>meet</u> to exchange them (but that wouldn't be possible!). We would mail each other (regular mail) the floppies. Actually, it would be enough when one of us receives the other's 'public token' so that he can securely e-mail his 'public token' fo rme to be able to e-mail him. (So, he should send me first). |
| P17 | <blank>  |
| P8  | use the generate-tokens function to make for ourselves a pair  |
| P30 | I would need to meet my friend to get their public token from them on floppy disc (-- inappropriate if we live so far apart). Alternatively we could rely on a mutually trusted friend who already has the public token of the friend I wish to write to and who has my public token, so can send me a secure e-mail containing the public token I require.  |
| P11 | 1) Once we had exchanged tokens, a) I would use her P. token to make unreadable the mail, b) I'd use my S. token to make-tamperproof, c) She'd use her secret token to make readable the received mail, d) She'd use my P. token to check identity of tamperproofing.  |
| P13 | First, since we live far away, I would have to get his public token from a common friend whose public token we both know. And he will get my public token from that friend. We both use tamperproofed messages.  |

### Staged (Lime) variant

|     |   |
|-----|---|
| P10 | Because my friends now live on the other side of the country, I can't trade the public token with them in person. So I have to trade the token via another person who I already have his public token. Let's see I choose Mr. A and Mr. A know my friend's public token. Then Mr. A send me a message which he does an make-tamperproof function in stating that he is sure that included public token belongs to my friend. Then since it was tamperproof, I can be sure the public token included belongs to my friend. |
| P4  | same as previous question, I'll probably just d/L his public token from the web.  |
| P5  | The best way would be to get the token from them in person, i.e. on a floppy. Other ways would be to get it from his/her web page (one that I've visited before) or have it emailed to me with some sort of note or message which verifies that it is my friend. -- also make sure that it's from the correct email address.  |
| P42 | I would send him an email telling him to write things in it to prove its himself then send me his public token, so that I could make the following private message unreadable.  |
| P33 | <del>We would need to exchange those pub</del> I would need to obtain my friend's private token. Either he & I would have to visit, send it through the mail, or use a mutual friend to transfer a copy of this token.  |
| P37 | For a low level of security, I could email my public token to my friend. For higher security I should send the token by snail mail. Also would include some info that would give my friend confidence that it was me.   |
| P16 | Yes, my friend would need the matching make readable function to read the   |



|     |   |
|-----|---|
|     | email.  |
| P14 | See above.  |
| P7  | Since I know my friend's email address, my friend could send a tamperproofed message containing his/her token. I could then check that the email had not been tampered with using my friend's public token. |
| P40 | I would want this to be pretty secure, but she is far away. (1) send her my public token by e-mail. (2) have her send me her public token by e-mail.  |
| P28 | <blank>   |

## (8) Are there any comments you would like to make?

### Hidden key pair (SSL) variant

|     |   |
|-----|---|
| P9  | Make-readable and make-tamperproof commands should be conjoined somehow. If you want it to be unreadable + tamperproof until the right person gets it, it's simpler to make it one command. |
| P35 | If I use option (2), my other friends might be curious of why I need to send a secure message to my old friend.   |
| P6  | It makes no sense to meet with my friend, just to exchange our tokens since I can tell him orally the message I wanted to tell him.   |
| P34 | No  |
| P18 | <blank>   |
| P41 | <blank>   |
| P12 | <blank>   |
| P27 | <blank>   |
| P15 | No.   |

### PGP variant

|     |   |
|-----|---|
| P29 | --  |
| P32 | <blank>   |
| P36 | I think w/this kind of message, worrying about the public token-secret tokens would give me more of a headache than worrying about some stranger reading my mail.     |
| P38 | The process is kind of confusing and time consuming, if it takes too long, I would not use it.  |
| P31 | --  |
| P17 | <blank>   |
| P8  | no  |
| P30 | Not viable for friend's/acquaintances/business partners living far apart. Alternative strategy involves asking a favour of a third party, which may not be desirable. |
| P11 | Switching 'tokens' and different passwords in this manner can be confusing; not conducive if in a hurry/not thinking, etc.  |
| P13 | It will obviously take up more than 10 minutes for the whole procedure.   |

### Staged (Lime) variant

|     |  |
|-----|--|
| P10 | <blank>  |
| P4  | maybe in corporate security would I need to put more security into this.   |
| P5  | I didn't choose to make this tamperproof because it seemed the focus was on confidentiality rather than verifying my identity.   |
| P42 | <blank>  |
| P33 | Physically exchanging tokens seems to be quite an effort. I would rather do so electronically.   |
| P37 | <blank>  |
| P16 | This part is a little confusing as to when you need to use your security token. I'm not sure if my friend would need it to read, or if I would need mine to send it. I'm also unclear on if he needs my public security token to read the mail |
| P14 | In reality, I'd probably go with the telephone conversation route.   |
| P7  | --   |
| P40 | I would do this by floppy (exchange tokens) if she weren't so far away.  |
| P28 | <blank>  |

### Question 3

Your Internet Service Provider (ISP) sometimes needs to send you email messages telling you to make certain changes in your software. If a hacker was able to forge one of those messages to you, so that you believed it had come from your ISP, then he or she might be able to break into your computer.

(9) Would you, in real life, think it was worth putting in some extra time to make sure this message was secure, rather than simply trusting a regular email? If yes, how much extra time (in seconds, minutes, hours, or days) would you think it was worth?

### Hidden key pair (SSL) variant

|     |  |
|-----|--|
| P9  | I would spend up to days to make sure it was secure so they can't break into my computer or anyone else's. |
| P35 | Yes, I would be willing to invest some minutes (up to 2-3 minutes per message).                            |
| P6  | yes, 1 hour for first time, 10 sec for each message  |
| P34 | Yes. It would be worth 5 minutes.  |
| P18 | Yes, 15 min  |
| P41 | 10 minutes... but then I don't have a multitude of personal/private/important files on my home PC          |
| P12 | Yes, it's worth it. at least a minute or two.  |
| P27 | Very worth the time. Possibly I would spend a day or two if I really had to.                               |
| P15 | Yes. 10 sec.   |

### PGP variant

|     |   |
|-----|---|
| P29 | Yes! 10-15 min to make sure everything is ok.   |
| P32 | Yes. 20 minutes.  |
| P36 | Yes, I would want this message secure. I don't want any hacker into my computer. I would say this is worth about 15 min extra time to secure the email. |
| P38 | Yes, a day or two.  |
| P31 | I would put an extra 1 hour at most.  |
| P17 | Yes, days.  |
| P8  | Yes, half an hour   |
| P30 | Yes. Half an hour.  |
| P11 | Yes, this is worth a few moments (up to ~5 minutes)   |
| P13 | Yes, 5 hours.   |

### Staged (Lime) variant

|     |  |
|-----|--|
| P10 | Yes. It can be worth a week (7 days) to make sure I get the correct message.   |
| P4  | I'd make it more secure. Probably 10-15 minutes.   |
| P5  | Yes: setup (1 time) - 1 hour max. time taken for each individual email: 30 seconds max.  |
| P42 | Yes, 2 min.  |
| P33 | Yes. A few minutes (< 30) to set up and then < 1 min every time I use that feature.  |
| P37 | I would not bother to use secure email here.   |
| P16 | Yes, 15 minutes  |
| P14 | Yes: hours.  |
| P7  | Yes. If I had to I would spend days setting up a system to make sure the emails I was receiving were valid, and not tampered with.         |
| P40 | yes. since there is a lot of important stuff etc on my computer I would take a lot of effort - a few days if needed - no more than a week. |
| P28 | yes 1 minute   |

(10) If you answered “yes” to question 9, then can you tell, from the software description you were given, which tokens and which functions you and your ISP would each need to use? If yes, please list them.

### Hidden key pair (SSL) variant

|     |   |
|-----|---|
| P9  | You would use all the functions. Make-unreadable, tamper-proofing, putting the tokens on disk and then passing them along in a secure way.  |
| P35 | My ISP would use the make-unreadable function and will use my token when creating the message. It will also use the make tamperproof function with its own token. I would use the check-tamperproofing function to verify the mail was sent by the ISP, so I need the ISP's token. I would also use the make-readable function with my token to be able to read the mail. |

|     |  |
|-----|--|
| P6  | <u>Me</u> - my token and ISP's token - make-readable - check-tamperproofing  <br><u>ISP</u> - iden - make-unreadable - make-tamperproof  |
| P34 | I would need to use their token with check-tamperproofing (and my token with make-readable)  |
| P18 | You need your ISP's token. They need to use make-tamperproof and you need to use check-tamperproof.  |
| P41 | - The ISP would have to set the "make unreadable" (using my token) + "make tamperproof" (using its token) functions on the email. Then I would use the "make-readable" (using my token) + the "check tamperproof" (using its token). |
| P12 | ISP would use make-unreadable & make-tamperproof to send it to me. They would need my token to do so. I would need make-readable & check-tamperproof to read the message my ISP sent. I would need my token to do so.                |
| P27 | We would need to know each other's tokens to make my message unreadable and to have them know it hasn't been tampered with.  |
| P15 | make tamperproof, check tamperproof  |

### PGP variant

|     |   |
|-----|---|
| P29 | ISP -> make-tamperproof & make-unreadable, me -> check-tamperproof and make readable  |
| P32 | ISP - secret token, tamper proof. me - ISP's public token, check-tamperproof.   |
| P36 | tokens: public, secret. function: make tamperproof, check tamperproofing.   |
| P38 | I need to have ISP's public token and use check-tamperproofing. ISP need to have it's secret token & use make-tamperproof.  |
| P31 | generate-tokens (one secret token, one public token). I use my secret token to make-readable the message ISP sent me. ISP should use my public token to make it readable for me specifically. (I am not clear the ISP already has my public token or will I have to give it to them!) |
| P17 | Make-tamperproof and Check-tamperproofing. Also a secret and public token.  |
| P8  | need to use ISP's public token to see if it was tampered with   |
| P30 | ISP must "tamperproof" its messages to me using <u>its secret token</u> and might also "make unreadable" to everyone but me using <u>my public token</u> . To "check tamperproofing" I need <u>the public token of the ISP</u> .  |
| P11 | Yes, my ISP would need to know my <u>public token</u> to make mail to me unreadable, and <u>their</u> s. token to make-tamperproof it. I would, in turn, use my s. token to read message and their p. token to check tampering.   |
| P13 | my pair of tokens, my ISP's pair of tokens. functions: make-unreadable, make-tamperproof, make-readable, check-tamperproof  |

### Staged (Lime) variant

|     |   |
|-----|---|
| P10 | ISP: use its own secret token to do make-tamperproof on the email he is going to send me. Me: use ISP's public token to do check-tamperproof function when I get the email. |
| P4  | use my public token & make-unreadable and use their secret token to make it make-tamperproof. then I'd use their public token to do check-tamperproof.                      |
| P5  | tokens: ISP would need to have my public token, and their own private.  |

|     |  |
|-----|--|
|     | I'd need their public token, & my private. functions: ISP would "make-unreadable" with my token and "make-tamperproof" with their own token. I would "make-readable" with my token, and "check-tamperproofing" with their token. |
| P42 | The ISP would need my public token to make-tamperproof then I would read the message with my private token.  |
| P33 | I would only need my own private token. The ISP would need a copy of my public token so it could be used to "make tamperproof" the message. I would use my private token to "check-tamperproofing).                              |
| P37 | <blank>  |
| P16 | security token, make tamperproof function, check tamperproof function, public token, make unreadable, make readable  |
| P14 | At least "make-tamperproof", but in this case <u>make-unreadable</u> would be an added check or precaution.  |
| P7  | The 'make tamperproof' function would be most useful here. My ISP would 'make tamperproof' using their secret token, then send me the public token to check that the message had not been tampered with.                         |
| P40 | ISP - needs a secret token. me - I need my ISP's public token. function - make tamperproof, check tamperproofing.  |
| P28 | I would need to use their make-readable + check tamperproofing with their public token. They would need to have used make-unreadable + make-tamperproof with their secret token.   |

(11) If you answered "yes" to question 9, then can you tell, from the software description you were given, what steps you and your ISP would each need to take in order to get those tokens at an appropriate level of security? If yes, please list them.

#### Hidden key pair (SSL) variant

|     |  |
|-----|--|
| P9  | Putting them on disk, with a private password and then passing it securely to the receiver.  |
| P35 | When subscribing to the ISP, the ISP will send me a floppy with their token. I will then send them my token through an e-mail with the make-unreadable function.   |
| P6  | - the ISP should send its token in physical form by secure express mail - once I've got its token I can send him my token by "secure" email  |
| P34 | * If they don't care whether anyone can read the message, i.e. if it is general FYI stuff, then they needn't do *. So if I could get their token from someone else using the same ISP then this could work.  |
| P18 | You'd probably have to gauge the likelihood of this threat mentally. It's probably real low. Your ISP would have sent you a token when it setup your account, either by email or possibly in the mail. Then all you'd do is use this token to check-tamperproof when you get this sort of message from your ISP. Threat is probably unlikely though. |
| P41 | I would probably want to go to the offices of the ISP to drop my token off + get one of it's.  |
| P12 | I would get my token by choosing a pass phrase with the security software. Then I would put my token on a disk and give it to the appropriate person at the ISP.   |
| P27 | If I could I would go to my ISP. If not I would probably call them.  |

P15 No.

### PGP variant

- P29 The best way would be to exchange them by person or by mail if it's relatively close.
- P32 ISP needs to use their secret token to tamper proof the email first. Then after I receive the mail, I use the ISP's public token to check-tamperproof and see if it is the ISP who tamper proofed the message last.
- P36 No, I think this information is lacking.
- P38 We should meet in person and ISP should create token before hand.
- P31 ISP sends me a public token (their public token for individual correspondence) I trust them that it belongs to a certain ISP representative. I e-mail to that representative my 'public token'. They e-mail me with that public-token.
- P17 - My ISP would need to use its secret token. - I would need to use my ISP's public token.
- P8 need to exchange public tokens and ISP need to use secret token to make-tamperproof it
- P30 A representative of the ISP and I would have to meet to exchange public tokens on floppy disc or choose a trusted intermediate who has both public tokens and of whom both the ISP and I already have the public token to transmit the information by email.
- P11 1) ISP obtains my p. token, 2) ISP sends mail to me, made unreadable (using my p. token), 3) ISP uses its s. token for anti-tampering, 4) I receive mail; use s. token to make readable, 5) I use their p. token to check tampering status.
- P13 Exchange public tokens. My ISP uses my public token to make-unreadable and his secret token to make-tamperproof. I use my secret token to make-readable and my ISP's public token to check-tamperproof.

### Staged (Lime) variant

- P10 ~~I have two ways. First (1) put my public token in a on floppy disks and meet ISP I still need to use the third side to trade the token with ISP (If I can meet him in person I,~~ I have two options. First (1) put my public token on floppy disks first time when I meet my ISP and get ~~their provider~~ its public token in same way. Or I can use the third side to trade the token. If I have a person A's public token and he knows for sure the ISP's token, then A will send me a message on which he does tamperproof function. When I receive it I use A's public token to do check-tamperproof function to make sure the ISP's public token given by A is not tempered.
- P4 we would d/L each other's token off the web
- P5 I think the most workable way would be to download it from the ISP's homepage. www.myisp.com since all ISP customers \*should\* know the web page of their own ISP.
- P42 I would give them my public token thourgh email ~~I make tamperproof w/ my private token. They would~~
- P33 I would need to give my public token to a representative of ISP (in physical form). Presumably the ISP & I would verify each other's identity during the exchange.
- P37 <blank>

|     |  |
|-----|--|
| P16 | My ISP server would have to use my public token, so that only I could read it. I would need my security token so I could read it. They would have to use their secret token to make the email tamperproof, and I would have to use their public token to check-tamperproof to make sure no one has tampered with it between them and me. |
| P14 | I would contact my ISP on the phone, and set up a token-pair, ground mailing my public token to them on diskette while requesting they do the same with their public token to me, and in the future only send update requests in unread, untamper secure formats.  |
| P7  | The best way would be to get a floppy disc with the token on it from the ISP, preferably in person from their office or something. (I don't know much about ISPs, do they have offices?) If this wasn't possible, the next best thing would be sending a tamperproofed message containing the token.                                     |
| P40 | I would like to get a floppy disk with the public token if possible. <del>If not, I would like to</del>  |
| P28 | They could have included their public token to me in the original written literature. Or maybe I'd get it from a friend who had used it successfully.  |

(12) Are there any comments you would like to make?

Hidden key pair (SSL) variant

|     |   |
|-----|---|
| P9  | <blank>   |
| P35 | I am concerned if the token I have from the ISP is also the same that everybody else has.   |
| P6  | <blank>   |
| P34 | <blank>   |
| P18 | <blank>   |
| P41 | <blank>   |
| P12 | To ensure maximum security, the disk I give the ISP would have to be locked away where only the responsible person could access it. |
| P27 | <blank>   |
| P15 | No.   |

PGP variant

|     |  |
|-----|--|
| P29 | --   |
| P32 | <blank>  |
| P36 | <blank>  |
| P38 | It might take a very long process to meet ISP.   |
| P31 | The first e-mail the ISP sends me is an unsecure one, since the proceeding correspondence depends on 'trust', I would expect to receive a 'trustful' e-mail, provided by interface, etc. |
| P17 | <blank>  |
| P8  | <blank>  |
| P30 | Sounds unworkable.   |

|     |  |
|-----|--|
| P11 | seems practical for this purpose, but 2 tokens is still an oddly complex method for communicating. |
| P13 | No   |

### Staged (Lime) variant

|     |   |
|-----|---|
| P10 | <blank>   |
| P4  | <blank>   |
| P5  | <blank>   |
| P42 | <blank>   |
| P33 | It would make the most sense to give this token as I'm purchasing the software (at a store...not just downloading it with a fee).   |
| P37 | My ISP doesn't send me instructions on changing software, and if they did, I would know what the instructions did.  |
| P16 | I would say if you were making this a manual on the security system, you may want to add an example or two of the use of the tokens and functions so that people can understand it a little better.                                 |
| P14 | Normally, I would contact my ISP by telephone and ask them about any email that requested me to change settings, rather than do anything via email.   |
| P7  | Can you make a message both unreadable and tamperproof? The description isn't clear on that point. Also, couldn't a hacker provide a false public token and a false message that was tamperproofed using the hacker's secret token? |
| P40 | <blank>   |
| P28 | <blank>   |

### Question 4

You have started a small company, with about 30 employees, which is busy developing a new product line, the details of which must be kept secret from the public and from your competitors. Your employees all need to communicate regularly with each other by email to keep each other up to date on the product strategy and progress. You are hiring additional people at the rate of one or two per week, and the new people need to be integrated into the email communications as quickly as possible.

(17) Would you, in real life, think it was worth putting in some extra time to make these messages secure, rather than simply relying on regular email? If yes, how much extra time (in seconds, minutes, hours, or days) would you think it was worth?

### Hidden key pair (SSL) variant

|     |  |
|-----|--|
| P9  | Yes it has to in order to keep the product secure you should spend up to <u>days</u> .                         |
| P35 | Yes, for the initial setup I would spend up to 4-6 hours.  |
| P6  | yes, 1 hour for first use, 10 sec/message  |
| P34 | Yes. Personally it would be worth 1 minute for each message depending on subject. Setup would be worth 1 week. |



|     |   |
|-----|---|
| P18 | YES 1 DAY   |
| P41 | Yes - 20 minutes per message + token making time  |
| P12 | Yes, though since there are so many communicating with each other, sending so many messages, to ensure efficiency the process should ideally only take a few seconds per message. |
| P27 | Yes, days.  |
| P15 | Yes. 1 min.   |

### PGP variant

|     |  |
|-----|--|
| P29 | Yes! A couple of minutes.  |
| P32 | 1 hour   |
| P36 | I think a secure message in this case would be beneficial because it involves company secrets. An hour of extra time to insure security would be good. |
| P38 | Yes, a day.  |
| P31 | Days.  |
| P17 | Yes, days.   |
| P8  | I'd spend a week making them secure  |
| P30 | Yes. I have no experience of business finance but probably a number of working weeks.  |
| P11 | Yes, up to 10 min, but not willing to do these steps EVERY time email is used.   |
| P13 | Yes, 2-3 days.   |

### Staged (Lime) variant

|     |   |
|-----|---|
| P10 | Definitely. Maybe <del>1 hours.</del> Because one week's training is necessary.   |
| P4  | probably a couple hours or a few days since it's essential to make sure documents don't leak out.   |
| P5  | Yep. setup: 4 hours max. for each message: 10 seconds.  |
| P42 | Yes, 5 min.   |
| P33 | Yes. A day to set up, < 30 min to add a new employee and < 10 sec for them to use it each time (or they won't do so).   |
| P37 | Yes, I would be willing to spend a day or two to initially setup it up for the company. I would spend an hour or so to set up for each new employee. I would not want more than 5 min to be added to the time for sending each email. |
| P16 | Yes, about a week.  |
| P14 | Yes, days, in advance.  |
| P7  | Yes, although doing this would be rather difficult and would take days.   |
| P40 | Yes. As much as needed -- weeks if necessary.   |
| P28 | yes, 5 seconds/message  |

(18) If you answered "yes" to question 17, then can you tell, from the software description you were given, which tokens and which functions you and your employees would each need to use? If yes, please list them.

### Hidden key pair (SSL) variant

|     |  |
|-----|--|
| P9  | <del>make-unreadable, make-readable</del> , make-tamperproof, check-tamperproofing, setting token, password so everyone knew it and then distributing it through the web page.   |
| P35 | All the messages would be sent with the make-unreadable option. Each employee needs his own token and everybody else token.  |
| P6  | - everyone in the company's token - make readable/unreadable - make/check tamperproofing   |
| P34 | For employee A, In sending a message to Emp. B, A would need to use B's token to make-unreadable and A's token to make tamper---, B would need to use A's token to check-tamper--- and B's token to make-read---                               |
| P18 | The new employee needs to have the tokens from his "Need to Know" group. Use all 4 functions.  |
| P41 | Write a message to someone - secure it using his token - make it tamperproof using yours. Read a message - unsecure it using your token - check for tamperproofing using his.  |
| P12 | each employee would need a list of all tokens belonging to everyone currently employed at the company, as well as their own token. They would be using all four functions on every message which contained information about the product line. |
| P27 | I would use make-unreadable and make-tamperproof. Receiver would use make-readable and check-tamperproofing.   |
| P15 | make-unreadable, make-readable, make-tamperproof, check-tamperproof  |

### PGP variant

|     |  |
|-----|--|
| P29 | New Employees - Generate-tokens to receive token. send -> make tamperproof & make unreadable. receive -> check tamperproof & make readable.  |
| P32 | me - secret, public, employee's public tokens, tamperproof, check tamperproof. employee - secret, public, each other's public token, my public token, tamperproof, check tamperproof.  |
| P36 | tokens: public. function: make tamperproof, make unreadable.   |
| P38 | I should create tokens and use make-unreadable, employees should have public token to use make-readable and check-tamperproof.   |
| P31 | The firm (I) would generate-tokens (private & public token). Every employee individually would generate tokens (public & private).   |
| P17 | Make-unreadable and Make-readable. Make-tamperproof and Check-tamperproofing. Secret and public token.   |
| P8  | Everyone will be told to use the generate-token function within the company. The make-unreadable, make-tamperproof, and check-tamperproof function will always be used.  |
| P30 | The messages need to be tamperproofed and made unreadable to all but the destinee. The sender/s requires the public token/s of the recipient/s to do "make unreadable" and his/her/their own private token/s to "make tamperproof". The recipient/s require/s the public token/s of the sender/s to check the tamperproofing and their own private token to make readable. |

- P11 1) Each employee has her own s. token; and her own p. token -> made public to all. 2) s. tokens used to make mail readable, make tamperproof. 3) p. tokens used for make mail unreadable, check tamperproofing.
- P13 I would need to use: my public and secret token, the public tokens of my employees. They would need to use: their pair, the public tokens of the other employees and mine. Function: make-readable, make-unreadable, make tamperproof, check-tamperproof.

### Staged (Lime) variant

- P10 For me: For every important email I send, I'll use my employee's token to make unreadable function and when I get email from my employees I will verify this identity by using his public token to do check-tamperproofing. For my employees, for every important email they send to me, he/she use his secret token to do make-unreadable and make-tamperproof on it. When I receive them I use my secret token to do make-readable on it, and use their public tokens to do check-tamperproofing.
- P4 sender - use receiver's pub token - make-unreadable, use their own secret token - make-tamperproof. receiver - use sender's public token - check-tamperproof, use their own secret token - make-readable.
- P5 Everyone would need their own private token, and everyone else's public token. To send "make-unreadable" with recipient's public token and "make-tamperproof" with one's own private token. To read "make-readable" with one's own private token and "check-tamperproof" with sender's public token.
- P42 we would need all each other's public keys and use all 4 functions mentioned in the description.
- P33 Yes. We'd need to use both the tamperproofing & readability features so as the boss, I would need to maintain a library of public tokens that all employees would have access to for the purposes of sending email. They would use their private tokens to read it.
- P37 Everyone would use all the functions - generate-tokens first, then make-unreadable + make-tamperproof to send and make-readable + check-tamperproofing to receive. Also, everyone would be instructed to use good pass phrases and not to write them down anywhere.
- P16 generate tokens, security token, public token, make unreadable, make readable, make tamperproof, check tamperproof
- P14 Secret + public token for everyone, make-unreadable + readable definitely, advise make-tamperproof, check-tamperproof. In previous answers, I wasn't as detailed with this listing of functions as I should have been.
- P7 We would probably need to all use the 'make unreadable' function so that company outsiders would not be able to read our messages if they intercepted them.
- P40 all tokens + functions. all employees would need everyone else's public tokens + everyone would need a secret token. all functions - make readable/unreadable, make/check tamperproof.
- P28 Any sender needs to use make unreadable + make tamperproof with their private token. Any receiver needs to use make readable + check tamperproofing with the sender's public token.

(19) If you answered "yes" to question 17, then can you tell, from the software description you were given, what steps you and your employees would each need to take to get those tokens at an appropriate level of security? If yes, please list them.

#### Hidden key pair (SSL) variant

|     |  |
|-----|--|
| P9  | The first person would tamperproof the message passing it to the next with absolute sureness that it belonged to the first person, it then can be passed along to the web.   |
| P35 | When hiring a new employee, human resources will provide the new employee either with a floppy disk with all the tokens, or just give him his new computer with everyone's tokens already copied. Human resources would also send all the old employees a make-unreadable mail with the token of the new employee. |
| P6  | - every existent user must receive by secure email an updated list of token every time some new people must be integrated in the communication. - every new user must receive by hand on a floppy a list of token of all the user (himself included).  |
| P34 | One way would be to have lots of employees meet with each new hire to exchange tokens on labeled floppy. Less securely, you could do this through interoffice mail (we're all one happy family after all).   |
| P18 | You would exchange tokens with the employee physically (on a disk). Then you could transmit the token of his Need-To-Know group via email using make-unreadable and make-readable. Probably not more than 5 people in a particular product group.  |
| P41 | Some trustworthy person in the company would give out their token to new members - then these people could go to a web page and download the secure tokens of other people in the company.   |
| P12 | If I was the boss, I would tell everyone to give me in person their token on a disk. Then I would compile them in a list (along with my token) and hand them back out to all the employees on disks, so everyone would have the complete list of tokens.   |
| P27 | Meet with the people in person to exchange tokens.   |
| P15 | No.  |

#### PGP variant

|     |   |
|-----|---|
| P29 | If all employees are working in the same building definitely by person exchanging disks.  |
| P32 | When someone sends a message, (to all other employees, including me), they should use their own secret token to tamper proof it. Then when everyone receives the mail we use check-tamperproof by the sender's public token to see if it has been changed.  |
| P36 | protect a message w/public tokens of all workers inside company. Then only the workers will be able to read the files.  |
| P38 | Yes, handle in person and make sure no token had been used by non employer.   |
| P31 | The firm (I) e-mail my public-token to the new employee. He would e-mail me his public-token to me, Then I would e-mail him the 'public-tokens' of the rest of the people. And I would e-mail the rest of the people his 'public token' since I already have the other people's 'public tokens' individually. |
| P17 | I would need to use my employee's public token. My employee would need to use their secret token. I would need to use my secret token. My employee  |

|     |  |
|-----|--|
|     | would need to use my public token.   |
| P8  | and our public tokens can only be exchanged within the company, by having people come in and tell us themselves  |
| P30 | The employer would have to distribute floppy discs with his/her public token to all new employees, and receive from each new employee a floppy disc with the employee's public token.  |
| P11 | 1) Each employee's p. tokens make public, known (to company). 2) Employees use p. token to make mail unreadable. 3) Employees use their s. tokens to tamperproof. 4) You (receiver) use your s. token to make readable. 5) You (receiver) use their p. token to check for tampering. |
| P13 | Create the token pair and secure it. The employees create their own token pair and secure it. Exchange public tokens.  |

### Staged (Lime) variant

|     |   |
|-----|---|
| P10 | Each of us put our own public tokens on floppy disks and trade them in the first-week training program.   |
| P4  | use a local intranetwork to distribute the employee's public tokens. that way, no one from the outside can access the local network.  |
| P5  | Using the 2nd method through a trusted 3rd party. For instance the boss (me) could send everyone a tamperproofed message with the public token(s) of new employee(s).   |
| P42 | We would exchange them by using discs. The new workers would get discs w/everyone's token (public) on them <del>and would then send everyone else a I</del> would get their public tokens and send a tamperproofed email from to everyone else in the office!   |
| P33 | Yes. When they're hired, I would create a token pair for them. I would retain a copy of their public token (and private, too?) and give them a disk with an up to date list of everyone else's public tokens.   |
| P37 | All employees would need to have everyone else's public token. Public tokens could be exchanged in person, since presumably everyone works in the same place.   |
| P16 | You would have to use the generate tokens function for each new employee. When sending an email, you will have to use that person's public token to make the email unreadable, and the people who are receiving the email have to use their security token to make it readable again. The person sending the email would have to use their security token to make the email tamperproof and the people who are receiving the email has to use the check-tamperproof function and know the public token of the person they are receiving the email from. |
| P14 | On at least a weekly basis, I think one designated "security officer" would generate secret + public tokens for all employees, possibly generating the passwords for the secret tokens as well. That person would distribute all public tokens + the specific secret token to each employee, and "archive" in some secure, offline place old token pairs.   |
| P7  | The best way to be both maximally secure, and to get new employees integrated as soon as possible would be to make a master disk of all employees' tokens and copy them for new employees. I could then send the <u>new</u> employees' public tokens to the old employees using 'make-tamperproof'.   |
| P40 | I would give my public token on a floppy to each employee + then send a tamperproof message with all the other public tokens. I am not sure of a secure enough + efficient way to do this. The person would have to give everyone his public token too which would be time consuming.   |

P28 Put public tokens at bottom of emails you send.

## (20) Are there any comments you would like to make?

### Hidden key pair (SSL) variant

P9 <blank>

P35 <blank>

P6 <blank>

P34 No

P18 <blank>

P41 I don't know how well this system would work after the company got beyond a certain size - it seems to cause logistical problems.

P12 yes, several. 1) the functions should be automatic to save time. i.e., when sending a message (with token) it should be automatically made unreadable and tamperproofed. when reading a message (with a token) it should be automatically made readable and check tamperproofed. 2) functions should be disabled for nonessential messages. 3) this whole system becomes more & more difficult to keep secure based on 2 factors - a) the increasing numbers of people who need each other's tokens, and b) the frequency of people joining & leaving the company. Probably the best solution to this is to 1) change everyone's tokens at random intervals and 2) change everyone's tokens whenever an employee leaves the company.

P27 <blank>

P15 likewise, I don't really understand this question.

### PGP variant

P29 --

P32 Since there shouldn't be private messages sending between the employees, I won't let them use make-unreadable. However, there is the problem of being seen by public/competitors.

P36 <blank>

P38 It is not that secure, each person need to have their own token because the information could get loss to non-company people.

P31 No.

P17 <blank>

P8 <blank>

P30 This might work.

P11 \*\* strong chance of public tokens being incorrect/complications arising. -- Many people! -> = delays in production. \* P. tokens -- necessary? Sound like overly complex user id's/email addresses.

P13 No.

### Staged (Lime) variant

P10 <blank>

P4 no.

|     |  |
|-----|--|
| P5  | I was thinking that in a small company situation, you'd want to send 1 message to multiple recipients. The security software description doesn't address this. Would the software take multiple recipients and encrypt & send it for each person? or would I need to do it manually. |
| P42 | <blank>  |
| P33 | What would I do if someone left? <u>Reassign all the tokens?</u> yuk. What about separation of work & personal email?  |
| P37 | <blank>  |
| P16 | You would have to make the company public tokens readily accessible in order to keep communications up. Is there any way to have the public token attached to an email so as to save time?   |
| P14 | In this situation, I would look into an off-line, in house server as a better, long-term solution.   |
| P7  | --   |
| P40 | <blank>  |
| P28 | <blank>  |

## Question 5

You are involved in a fairly high-stakes lawsuit, and you need to communicate regularly with your lawyer about it by email. You have told your lawyer things about the circumstances of the suit that you expect to be covered by attorney-client privilege, and you need to discuss those things in your email messages.

(13) Would you, in real life, think it was worth putting in some extra time to make these messages secure, rather than simply relying on regular email? If yes, how much extra time (in seconds, minutes, hours, or days) would you think it was worth?

### Hidden key pair (SSL) variant

|     |  |
|-----|--|
| P9  | It is very important to keep this secure, so it's worth spending up to days.                               |
| P35 | It is worth a couple of hours for the initial (one time only) setup. Then a couple of seconds per message. |
| P6  | yes, 1 hour/first time, 10 sec/per message   |
| P34 | Yes. It would be worth 10 minutes.   |
| P18 | DAYS! Would have to be set up or would not use. Maybe not even use it anyway.                              |
| P41 | Yes -> probably 20 minutes for a message (+ the original token making + exchanging time)                   |
| P12 | yes, 1-2 minutes per message   |
| P27 | Security is very important here. Days if necessary.  |
| P15 | Yes. 20 sec.   |

### PGP variant

|     |  |
|-----|--|
| P29 | Yes! 15-20 min.  |
| P32 | yes, 1 hour  |
| P36 | Definitely need security with this case. I think it's worth at least a good 1 hour to get message secured. |
| P38 | Yes, few days.   |
| P31 | Days.  |
| P17 | Yes, days.   |
| P8  | I'd spend a couple of days doing that  |
| P30 | Yes. A week of evening -- 20 hours +   |
| P11 | Yes -- few minutes, but <u>NOT</u> each time I read/sent mail -- a hassle.                                 |
| P13 | Yes, 1 day.  |

### Staged (Lime) variant

|     |  |
|-----|--|
| P10 | Definitely. Even more than 10 days.  |
| P4  | yes. 20 minutes.   |
| P5  | Yes: setup - 3 hours max. time for each message - 3 minutes max.   |
| P42 | Yes, 5 minutes   |
| P33 | Yes. An hour or two to set up and then <15 min each time it's used.  |
| P37 | I would be willing to spend up to an hour to secure the email. I would be willing to spend an extra 5 min each time I sent an email. |
| P16 | Yes. 15 minutes.   |
| P14 | Yes. <u>Years.</u>   |
| P7  | Yes. I would spend days to make a system that would make my emails secure.   |
| P40 | Yes. a few days jus tlike the other. no more than a week.  |
| P28 | yes 1 minute   |

(14) If you answered “yes” to question 13, then can you tell, from the software description you were given, which tokens and which functions you and your lawyer would each need to use? If yes, please list them.

### Hidden key pair (SSL) variant

|     |  |
|-----|--|
| P9  | All the functions described, make-unreadable, make-readable, make-tamperproof, check-tamperproofing, setting a password, and then passing the tokens on in a secret way.   |
| P35 | I would use the make-unreadable and the make-tamperproof functions. I need my token and that from my lawyer. My lawyer would use the make-readable and check-tamperproof function. He needs both my token and his token. |
| P6  | Me - My own token - My lawyer's token - make readable/unreadable - make/check tamperproofing   Lawyer - his/her token - My token - make readable/unreadable - make/check tamperproofing                                  |
| P34 | In sending him stuff, I'd need to use his/her token with make-unreadable and my token with make-tamperproof. In receiving stuff, I'd need to use   |



|     |   |
|-----|---|
|     | his token with check-tamper--- and my token to make-readable.   |
| P18 | You would physically exchange the tokens. Both ends would use make-unreadable and make-readable.  |
| P41 | If I wrote a message I would use the lawyer's token to make it unreadable, my token to tamperproof it. Then he or she would have to use his/her token to make it readable + mine to check the tamperproofing. |
| P12 | My lawyer would both need to use make-unreadable & make-readable, make-tamperproof & check-tamperproof. We would need each other's tokens as well as our own.   |
| P27 | Exchange tokens to make messages unreadable and to know they haven't been tampered with.  |
| P15 | make unreadable, make-readable  |

### PGP variant

|     |   |
|-----|---|
| P29 | sending -> make-tamperproof & make-unreadable. receiving -> check-tamperproof & make-readable.  |
| P32 | me - secret token, public token, other's public, make-readable, unreadable, check-tamperproof, tamperproof. lawyer - same.  |
| P36 | get from the lawyer his public token and make it unreadable w/that public token. token: public token. function: unreadable.   |
| P38 | Yes, both of us should create token, use make-unreadable, make readable, make-tamperproof, check-tamperproof.   |
| P31 | Both sides generate-tokens & have a pair of private & public tokens.  |
| P17 | Make-unreadable and Make-readable. Make tamperproof and Check-tamperproofing. Secret and public token.  |
| P8  | we both need to use out secret & public tokens, exchange public tokens in person. both use make tamperproof functions and make-unreadable functions for the emails we send to each other. and I'd use the check-tamperproof using his public token. |
| P30 | We would each need the other's public token, in addition to our own secret token, so as to both "make unreadable" and "tamperproof" our email.  |
| P11 | 1) My p. token - used by lawyer to send unread. mail. 2) Lawyer's p. token - to check tampering status. 3) My s. token - to make-readable mail. 4) Their s. token - to make tamperproof.  |
| P13 | my pair of tokens, my lawyer's pair of tokens. We would use all the functions described in the software description.  |

### Staged (Lime) variant

|     |   |
|-----|---|
| P10 | For me: (1) I use my lawyer's public token to do make-unreadable. (2) I also use my secret token to do make-tamperproof on it. (3) After I receive the email from my lawyer, I use my secret token to do make-readable and use my lawyer's public token to do check-tamperproofing on it to make sure it hasn't been forged. For my lawyer, he does exactly things that I need to do. |
| P4  | my lawyer - use his secret token to make msg make-tamperproof, use my public token - make unreadable. me - use his (lawyer) public token - check-tamperproof, use my secret token to make-readable.   |
| P5  | me: I need the lawyer's public key*, and my own private key. When sending I'd "make-unreadable" with his *key and "make-tamperproof" with my key. When reading his messages, I'd "make-readable" with my token and "check-  |

|     |  |
|-----|--|
|     | tamperproofing" with his public token. Lawyer: same thing as me, just switching roles. (* key = token)   |
| P42 | We would both need each others public tokens. We would use make-(un)readable + make-tamperproof + check tamperproof functions.   |
| P33 | My lawyer and I would need to exchange our public tokens (perhaps even make a set just for the purposes of this lawsuit). We would use the public tokens to "make-unreadable" and the private tokens to "make-readable"    |
| P37 | We would both use generate-tokens and exchange public tokens. When sending email, we would both use make-unreadable and make-tamperproof. We would use make-readable and check-tamperproofing when receiving email.        |
| P16 | make unreadable, security token, public token, make-readable, make tamperproof, check tamperproof  |
| P14 | Make unreadable, make "", make tamperproof, check "", generate tokens.   |
| P7  | We would most likely use 'make unreadable'. I would use my lawyer's token ot make the message unreadable, and my lawyer would use his/her secret one to read it, and vice versa.   |
| P40 | I would like to make everything secure due to the situation (high-stakes lawsuit). tokens - each have each other's public. each have a secret token. functions - all (unreadable/readable) (tamperproof) check tamperproof |
| P28 | Before sending, we would each use make unreadable + make tamperproof using private tokens. To read, we would each use make readable + check tamperproofing with the other's public token.                                  |

(15) If you answered “yes” to question 13, then can you tell, from the software description you were given, what steps you and your lawyer would each need to take to get those tokens at an appropriate level of security? If yes, please list them.

#### Hidden key pair (SSL) variant

|     |  |
|-----|--|
| P9  | To hand them over in person only.  |
| P35 | We will personally exchange floppy disks with the tokens.                  |
| P6  | Meet "physically" and exchange our tokens                                  |
| P34 | We could meet and exchange tokens on floppy.                               |
| P18 | You would physically exchange the tokens. You would exchange floppy disks. |
| P41 | I would meet w/my lawyer + trade disks containing tokens.                  |
| P12 | Meet in person & exchange tokens on disk.                                  |
| P27 | Go visit w/my lawyer to swap tokens.                                       |
| P15 | No.  |

#### PGP variant

|     |  |
|-----|--|
| P29 | By person! You would eventually meet your lawyer in person.  |
| P32 | yes. before each of us sends the message, we should use the other person's public token to make unreadable, then also use our own secret token to make tamperproof. After we receive mail, we use our secret tokens to make readable and use the others public token to check-tamperproof. |
| P36 | yes. I think I answered question above.  |

|     |  |
|-----|--|
| P38 | Yes, we should meet in person to trade tokens.   |
| P31 | The attorney sends me his public token <u>in a floppy disk</u> . Then with his public token, I send him my public token (via e-mail).  |
| P17 | - I would need to use my lawyer's public token. - My lawyer would need to use his secret token. - I would need to use my secret token. - My lawyer would need to use my public token.  |
| P8  | exchange public tokens in person after we each made up our own tokens  |
| P30 | Meet in person to exchange floppy discs containing our public tokens.  |
| P11 | 1) Lawyer gets my p. token, sends unread. mail to me. 2) Lawyer uses his s. token to tamperproof. 3) I use my s. token to make readable. 4) I use their p, token to check for tampering.   |
| P13 | Create my pair, choose a pass phrase, so that I can access my pair only with this phrase. Make a back-up copy of pair and store in a safe place. Same procedure for my lawyer. Exchange public tokens in person or using a third person that uses the software and we know his public token and he knows ours. |

### Staged (Lime) variant

|     |   |
|-----|---|
| P10 | We can put our own public token on floppy disks and trade them in person. Or via one of my friend and he is also the client of my lawyer. If I know my friend's token. And he sends me an tamperproofed email to tell me my lawyer's token. And he can also give my public token to my lawyer in the same way.  |
| P4  | I'd d/L from a "secure server" on the web if possible or else probably exchange token in person.  |
| P5  | I'd say the floppy disk would be the way to do it.  |
| P42 | I would exchange discs w/our public keys on them.   |
| P33 | Yes. When we meet, we exchange our public tokens on disk.   |
| P37 | For ideal security, we should meet in person and swap public tokens. It also might be sufficient security to exchange tokens via email.   |
| P16 | Whoever is sending the email would have to use the other person's public token to make it unreadable. The person would then need to use their security token to make it readable. The person sending the email would have to use their security token to make the email tamperproof. The person receiving the email can check-tamperproof by using the check tamperproofing function and by knowing the public token of the person sending the email. |
| P14 | I would want to generate a new set of tokens for each separate day if not each individual message, but no more than 3 or 4 tokens in advance. Once a week, I would exchange public tokens with my lawyer on diskette, with a verbal understanding on what order they would be used in. I would physically destroy the diskettes once the tokens had been used.  |
| P7  | Meeting in person, and conducting a floppy disk exchange of public tokens would make the most sense from a security standpoint.   |
| P40 | definitely exchange public tokens by floppy and meet in person  |
| P28 | I would ask him to write it down.   |

(16) Are there any comments you would like to make?

Hidden key pair (SSL) variant

|     |  |
|-----|--|
| P9  | <blank>  |
| P35 | The reason I use both functions is to make sure my lawyer cannot alter the contents of my mail after he opens it.                        |
| P6  | <blank>  |
| P34 | No   |
| P18 | If someone can get access to your hardware or his, No scheme will work. You'd need to exchange verbal passwords to augment this as well. |
| P41 | This is pretty private stuff - I don't know if I'd trust it to email even if I was pretty certain the system was secure.                 |
| P12 | Again, disks would need to be locked where only my lawyer & I could get to them.   |
| P27 | <blank>  |
| P15 | I really don't understand the question before (#15).   |

PGP variant

|     |   |
|-----|---|
| P29 | --  |
| P32 | <blank>   |
| P36 | <blank>   |
| P38 | This software would be useful in this situation.    |
| P31 | --  |
| P17 | <blank>   |
| P8  | <blank>   |
| P30 | <blank>   |
| P11 | See above (#13) - Bad for frequent, quick messages. |
| P13 | No.   |

Staged (Lime) variant

|     |  |
|-----|--|
| P10 | Maybe trading token is a little bit troublesome. After all, it can't be as easily as exchanging email address. |
| P4  | no   |
| P5  | <blank>  |
| P42 | <blank>  |
| P33 | Does the tamperproof feature let you know if the email has been read by a third party?                         |
| P37 | <blank>  |
| P16 | <blank>  |
| P14 | In <u>reality</u> , I would <u>never</u> discuss <u>anything</u> with my lawyer via <u>email</u> .<br>Period.  |
| P7  | ---  |
| P40 | <blank>  |

## Debriefing Questionnaire

1) Do you think the description provided the right amount of information about how the security worked? (circle one)

- (a) Not nearly enough information
- (b) Almost enough information
- (c) Just the right amount of information
- (d) Slightly too much information
- (e) Way too much information

### Hidden key pair (SSL) variant

|     |   |
|-----|---|
| P9  | b |
| P35 | b |
| P6  | b |
| P34 | b |
| P18 | b |
| P41 | c |
| P12 | c |
| P27 | b |
| P15 | b |

### PGP variant

|     |   |
|-----|---|
| P29 | b |
| P32 | c |
| P36 | b |
| P38 | b |
| P31 | c |
| P17 | c |
| P8  | b |
| P30 | c |
| P11 | b |
| P13 | c |

### Staged (Lime) variant

|     |   |
|-----|---|
| P10 | c |
| P4  | d |
| P5  | c |
| P42 | c |

|     |   |
|-----|---|
| P33 | b |
| P37 | b |
| P16 | b |
| P14 | c |
| P7  | b |
| P40 | c |
| P28 | c |

## 2) What was the most confusing part of the description?

### Hidden key pair (SSL) variant

|     |  |
|-----|--|
| P9  | trading the tokens (the second way)  |
| P35 | It wasn't clear when I should use one option over the other, or if I could use both functions simultaneously (which I did assume). |
| P6  | the "protecting your security token" part  |
| P34 | Security functions   |
| P18 | Exchange of token with trusted 3rd party.  |
| P41 | How security tokens are traded   |
| P12 | I had to read the "security tokens" a couple times because the sentences seemed a little dense & convoluted.                       |
| P27 | the token part. Not sure if there is one or two or a token and a password.   |
| P15 | the token-thing is pretty confusing  |

### PGP variant

|     |  |
|-----|--|
| P29 | The second way to trade public token. Very hard to understand. Has to be more specific with examples.  |
| P32 | the second way of trading...   |
| P36 | generate tokens and the relationship of public + secret tokens.  |
| P38 | Who use what token to use function   |
| P31 | In 'Protecting your secret token' part: ...to choose a pass phrase (like a password, <u>but no longer</u> , since it has multiple words). I am not clear about this. |
| P17 | The description about the tokens took some time to understand and I still wasn't 100% sure if I understood them fully.   |
| P8  | People who don't know anything about the program must be told how the system must work, and each step to achieving those functions                                   |
| P30 | None   |
| P11 | Using public + secret tokens in same application, yet differing ways for receiving vs. sending mail --> had to reread.   |
| P13 | Which token to use for each function.  |

### Staged (Lime) variant

|     |   |
|-----|---|
| P10 | <del>Trading public token via Why is it danger</del> Trading public token via third party.  |
| P4  | extra words added in the paragraphs. in trading public tokens to get basic security.  |
| P5  | First page under 'security tokens' 2nd paragraph: "If a public token is used to do make-unreadable..." the "to do make <newline> unreadable" was confusing.   |
| P42 | the <u>names</u> of the functions + the method of obtaining tokens  |
| P33 | - unanswered questions... I expected the program to have more options/features.   |
| P37 | The section on how an attacker could trick people into using incorrect tokens. It's a little confusing to understand exactly how an attacker would intercept emails.                                      |
| P16 | how exactly the public tokens work, and is it like the email addresses now. that was not very clear.  |
| P14 | I think you need to emphasize that tokens are <u>only</u> secure if traded on diskette or otherwise "off-line". Mention this <u>first</u> , and emphasize all other methods (email, etc) can be "hacked". |
| P7  | The part about the tokens took me a while to understand. I'm still not sure what exactly a 'token' is. Is it a password? Some kind of program part?   |
| P40 | understanding what functions required which type of token   |
| P28 | I think a token is just an ASCII string, but am not sure. The whole token concept may be overly mysterious/vague.   |

### 3) What was the clearest part of the description?

#### Hidden key pair (SSL) variant

|     |  |
|-----|--|
| P9  | security functions   |
| P35 | what each function did                                     |
| P6  | the bulleted list in "security tokens" part                |
| P34 | Trading Security Tokens                                    |
| P18 | First part about generation of token, and the 4 functions. |
| P41 | Setting the passwords on your security tokens              |
| P12 | "security functions"                                       |
| P27 | make-unreadable and make-readable                          |
| P15 | what each function does                                    |

#### PGP variant

|     |   |
|-----|---|
| P29 | functions - make readable, make unreadable                      |
| P32 | the security tokens. the bullet points clarify everything well. |
| P36 | security functions  |
| P38 | Security functions  |

|     |  |
|-----|--|
| P31 | 'Trading public tokens' part.  |
| P17 | The explanation of each of the 4 functions.  |
| P8  | The number of functions that are available for security and the necessity of a secret and public token |
| P30 | The "bulleted" list of instructions  |
| P11 | Secret tokens -- most beneficial to your security.   |
| P13 | the security functions   |

### Staged (Lime) variant

|     |   |
|-----|---|
| P10 | Second part regarding how the security token works.                   |
| P4  | - protecting your own secret key. - security function.                |
| P5  | The purpose of the product  |
| P42 | the way the functions work, which function to use how                 |
| P33 | The bulleted list that summarized how to use the tokens.              |
| P37 | The security functions section (description of what the functions do) |
| P16 | the functions and how to use them                                     |
| P14 | How to use security tokens.   |
| P7  | Everything else was very easy to understand.                          |
| P40 | explanations of how to trade tokens                                   |
| P28 | It's well-organized. I do understand.                                 |

### 4) What question about the description would you most like answered?

#### Hidden key pair (SSL) variant

|     |  |
|-----|--|
| P9  | how do you post it on the web securely   |
| P35 | Can I use both options (make-unreadable and make-tamperproof) simultaneously?  |
| P6  | How can I recover my token from a backup copy  |
| P34 | It would have been nice if it more <del>clearly stated</del> specifically described how to handle the previous situation |
| P18 | How can trading security tokens be made secure?  |
| P41 | How are the security tokens used to secure information distributed on a database/web page?                               |
| P12 | wouldn't it even more secure just to <u>verbally</u> inform people of their token passwords?                             |
| P27 | how would you swap tokens with someone if they are across the country?   |
| P15 | how exactly does the receiving person use their token?   |

#### PGP variant

|     |   |
|-----|---|
| P29 | Second way to trade public tokens. Why is "pass phrase"?                  |
| P32 | If I want to send an email to multiple recipients, can I make it readable |



|     |   |
|-----|---|
|     | to only those recipients? Because in the description it says "only a specific person", what about persons?  |
| P36 | I think the lawyer question, in terms of security was the most important.   |
| P38 | Could you explain the part talking about public token transfer via tamperproofed message?   |
| P31 | Questions related to 'Trading public tokens'.   |
| P17 | If you wanted to protect the security of a message, I wasn't sure why you wouldn't always want to use the 4 security functions and the 2 tokens like the previous 5 question had asked for. |
| P8  | what if our password phrase for the secret tokens was forgotten, how do we find out and how can we make sure it's secure  |
| P30 | What constitutes a "safe" place to store the token pair backup (and what do you need it for?)   |
| P11 | -> Not practical: physically meeting a person to exchange public tokens... must be easier electronic method (published directory of p. tokens?)   |
| P13 | If I create my own token pair, do I know that the certain token is not already used by some other person?   |

### Staged (Lime) variant

|     |  |
|-----|--|
| P10 | How do I and my partners need to do to protect my message  |
| P4  | pretty clear for the most part. I understood the descriptions.   |
| P5  | If I have 1 message to send to 10 people all of whom I have a public key for, do I send it 10 times manually? or just once & let the software encrypt it 10 times for each person?   |
| P42 | An easier (secure) way to give someone a public token safely w/out physical contact  |
| P33 | Can you make multiple token pairs for the same person? How reliable is the system?   |
| P37 | What type of software would be used and would it be easy to use and stable/reliable. Also, could public tokens be sent by snail mail on paper (without using a disk).  |
| P16 | Why is trading public tokens such a security risk, the security token is used for most of the important tasks  |
| P14 | <blank>  |
| P7  | What exactly is a token? Can you make a <u>group</u> token pair for sending mass, but secure, emails to a group of people? If you do this, do you still retain the security? Also can you both make something unreadable <u>and</u> tamperproof? |
| P40 | Is there a safe way to give many people your public token and to get theirs on a regular basis (company question in previous task)   |
| P28 | what's a token?  |

5) Do you think you would make use of security software like this for your email messages if it was available? (circle one)

- (a) Not at all
- (b) Once in a while

- (c) Frequently
- (d) Almost always
- (e) Always

#### Hidden key pair (SSL) variant

|     |  |
|-----|--|
| P9  | b  |
| P35 | b  |
| P6  | b  |
| P34 | c  |
| P18 | b  |
| P41 | b (for important, private emails but I only write those once in a while! |
| P12 | b  |
| P27 | c  |
| P15 | a  |

#### PGP variant

|     |  |
|-----|--|
| P29 | b -> Don't have much need of secure email yet! |
| P32 | c  |
| P36 | b  |
| P38 | b  |
| P31 | d  |
| P17 | a  |
| P8  | b - in the future                              |
| P30 | b  |
| P11 | c --> If made less complex.                    |
| P13 | b  |

#### Staged (Lime) variant

|     |  |
|-----|--|
| P10 | b  |
| P4  | b only for very private emails & only if sniffing programs become more widely used |
| P5  | b  |
| P42 | b  |
| P33 | c  |
| P37 | b  |
| P16 | c  |
| P14 | c  |
| P7  | b  |
| P40 | b  |
| P28 | b  |

6) Do you have any other comments you'd like to make?

Hidden key pair (SSL) variant

|     |   |
|-----|---|
| P9  | <blank>   |
| P35 | It would be useful to provide some example in the description, as to make it easier to understand (or to decide) when you should use one function over the other.   |
| P6  | I think the last sentence of the first paragraph in "security tokens" section, would better go at the end of that section (on a line by itself)   |
| P34 | I wish I had read the last line of Trading Security Tokens more carefully. I can see how that would be useful. You can send a public message readable to all with your otken which has been tamperproofed with it. You then can include text which everyone knows is from you, including the token which can then be used to send unreadable messages to you. |
| P18 | If you set this scheme up ahead of time and the software automatically enforced the security restrictions, it would work. Otherwise, you'd probably forget about how it worked, and never take the time to use it. The likelihood of someone targeting me for this is unlikely.   |
| P41 | <blank>   |
| P12 | Just that the more people involved, the harder it is to maintain the same level of security using this system.  |
| P27 | <blank>   |
| P15 | Nope.   |

PGP variant

|     |  |
|-----|--|
| P29 | secret & public token system seems pretty clever but trading those public token might be a bit confusing.                                      |
| P32 | Overall efficient. Should verify that each person can only generate one pair of tokens?  |
| P36 | like I have mentioned over 90% of my email, I don't mind being read by others.   |
| P38 | This software is not very useful to me, but for some other situation, it is useful. But it takes a long time to understand and get used to it. |
| P31 | No.  |
| P17 | If I had to talk to someone about an important issue and was worried about security, I would call them and not e-mail them.                    |
| P8  | <blank>  |
| P30 | The floppy disc trading is a big problem.  |
| P11 | Public tokens seem, in some ways, to serve same function as your user id/password (i.e. to read mail <u>only</u> for you).                     |
| P13 | No.  |

Staged (Lime) variant

|     |   |
|-----|---|
| P10 | <blank>   |
| P4  | I forgot to use the generate-token function each time I want to make more |

|     |  |
|-----|--|
|     | of my own public tokens.   |
| P5  | <blank>  |
| P42 | nope   |
| P33 | see #2   |
| P37 | I think secure email is still very important. Email communication is outgrowing other forms of communication and there needs to be a way to secure it.   |
| P16 | If this is for general consumption, use laymen terms and examples. It would make it easier for most people to understand.                                |
| P14 | You might want you use "real world" situations to illustrate use of tokens: it may double length of readings, but help people to understand them better. |
| P7  | --   |
| P40 | no   |
| P28 | <blank>  |

## **APPENDIX C Metaphor comparison test materials and data**

This appendix contains test materials and collated data from the comparison of two variants of my visual metaphor set against the visual metaphors from PGP 5.0, as described in Section 6.1.2 of Chapter 6. The contents are as follows.

C.1: Reproductions of the three variant metaphor set presentations that were given to the test participants.

C.2: Questions given to participants, with collated responses, including debriefing questionnaire. The questions are phrased here as they were given for the Lime metaphor variants; the actual questions given for the PGP variant referred to functions rather than locks.

The screening questions, consent form, and initial briefing are not included in this appendix because they were the same as those for the previous test; see Appendix B for copies.

### **C.1 Presentation variants**

This section reproduces the three variant presentations given to the user. The first is the PGP metaphor set variant, the second is the basic Lime metaphor set variant, and the third is the extended Lime metaphor set variant, with signature locks and certification locks.

# YOUR ELECTRONIC MAIL SECURITY SOFTWARE

## Security functions

Your electronic mail security software provides functions for protecting your mail messages against unauthorized reading (eavesdropping) and unauthorized modification (tampering or forgery).



To protect a message against unauthorized reading, use the **encrypt** function on it. Then an authorized person will need to use the matching **decrypt** function in order to read the message, and no-one else will be able to read it at all.

```
-----BEGIN ENCRYPTED MESSAGE-----
qANQR1DBWU4D3aRT5uqDyhwQCACPaEYliP+nkQI+L90VhIY6QMSayQDTSX/nrPo
s3KUD90cyAeMOWtZAqICZ442qVzWq0u9cqC2elKeFYI8yQ3Z4wT4R+mSoYlCAgzc
tRsbJdNoMIyPesDNXt+gYXvQZ7eFOrOsRv2j5PNNkVGRaXgNXvzNNdhrDYj4dYRk
tw0n22KVKlgjAMhnyogoYXG3lPEp2atIh07hgNyGw+s09HbkNReONwdh3WuhZYpw
3jt99TUmjDR1sHF4eJyN5iNHRewMj9QLYsYJAYweYNrPqC HHU/3OeGj/Ks7PPgV
ezykK+XmF26gMcC6reYEzmU6xddgj4OvoV+bc7cWUhbNrJOVCACNAW+9z/NzH0NQ
ne8lJ4zyH8cTy5SvcG0dk7U76rMaC+3hIgsml9uU8G9NhuoVYKaBPPWXR56L9Hbs
PaF9S4LeQr7gusybyO54ghf2lUOVROUxqhOxn4EafI763k1VKaBD/cerMG/fWhh+
B15fVdknZKPLDFAgUa7Pdm6OF7sNtDjxpU7aXY7QakHZLHIE++U1x/7nSwj2mEBp
aUV9PZUYo/a+93qXlVq1vbXu+qQaq2Ec3qgrKQJkhp+Z50A00NKGP5ZnupZIM7jY
FvNa+rgcoA9t9iGYSxZQlWgTir/i4vZDDS3+SmZVu8QNr3ZWbpF1PMAMPzjb69yN
ukI5HLZxycHpOC/tJRsJ9sCd06lqDFNori17s5nR35i1GHqZRLsaa6BqdMikIiDk
ABg4tkOfn0ehnW+JVdM4XHZRdyouZ408ceAhiHOHQbWUTXdzfgrlUffn8GlnzYlg
-----END ENCRYPTED MESSAGE-----
```

### An encrypted message



To protect a message against unauthorized modification, including forgery, use the **digital signature** function on it. People who view the message will then be able to use the matching **verify-signature** function to see who signed the message and to verify that no later modification has occurred.

```
-----BEGIN DIGITAL SIGNATURE-----
iQEVAWUBodtEbXjichGj2qlzAQEm5QgApkzbziwAYh0/k0i5IHQpF8qPppiO0lth
qVRmtE9XzGwWcG1socEyJ2p5udasVaxapaSDws9QhrkiRW/8PY4EONm2ZbyW7qwe
GB0Y/LCL1WrbhrQkw9kXecDWH704QJTacmjuhOp+mPwpvQmR9CJp66RBCCQWfQRx
rxKkpKZuIRgTqtSRPDGwnNI2wTlOoe3FhQS290G/R/GE1xmb+eYmzwIk/qNKmrH8
O2Xp0AQnSd4M3JSWv+XNE7BK8gsSOrIm1ZbDRvdO2igwHByw8GU+qH3wOBxBf3Ij
IZMravzi6ygzN+ksXWbZ9wyl/g8rlhYEHo/UzDbcX1hfCrBSdOQNdg==
=IcGR
-----END DIGITAL SIGNATURE-----
```

### A digital signature on a message

Each of these four functions must be used with a **key**.

## Keys



Each person who uses the security software must have their own matched pair of keys, consisting of one **secret key** and one **public key**. Secret keys must be carefully protected so that only their owners have access to them. Public keys, however, are meant to be freely distributed and traded, so you should give your public key to everyone you send messages to and get their public keys in return. You can use the **generate-keys** function to make a key pair for yourself.

Secret keys are used for proving identity. They can be used this way while still remaining secret because of the special power of a matched key pair. If a public key is used to encrypt a message, then only someone with the matching secret key can make that message readable again. Likewise, if a public key can be used to do verify-signature on a message, then that proves that the message was signed with the matching secret key.

- To protect a message so that only a specific person can read it, use that person's public key to encrypt it.
- To read a protected message that has been sent to you, use your secret key to decrypt it.
- To protect a message so that people can be sure it hasn't been forged, or changed since you protected it, use your secret key to digitally sign it.
- To verify that no-one has changed a message that has been sent to you since it was signed, and to verify the identity of the person who signed it, use that person's public key to do verify-signature on it.

### Protecting your secret key



Because your secret key is your proof of identity, you need to carefully protect it against theft or loss. When you create your key pair, your software will ask you to choose a pass phrase (like a password, but longer, since it has multiple words). Your software will not let anyone access your secret key unless they have that pass phrase, so choose something that you will find easy to remember, but that other people won't be able to guess. Then make a backup copy of your key pair and store it in a safe place.

### Trading public keys to get basic security



The security of your messages depends on having the right public key for each person. If an attacker can trick you into thinking their public key belongs to someone you send messages to, then the attacker can read protected messages you send to that person, and forge digitally signed messages from that person to you. When you trade public keys, you need to take precautions to make sure you aren't being tricked.

The simplest way to trade public keys is usually to send them in mail messages or put them up on personal web pages for downloading. The risk is that an attacker could set up

a fake web page or forge an email message so that it appears to be from someone you know. For basic security, protect yourself against these kinds of tricks by asking common sense questions. Have you been to this person's web page before, and is it at a web address you know that person uses? Does the message with the key in it sound like that person, and mention things that person would know? Does it come from an email address that you know that person uses? Likewise, when you send your public key to other people, include a note that will help them be sure the message came from you.

This level of security is enough to protect your messages against random eavesdropping and simple forgery, and against attackers who are looking for general vulnerabilities and have no reason to work hard to target your messages in particular. If your messages contain very sensitive or valuable data, or if you have some other reason to think an attacker might want to single you out as a target, then you should consider a stronger level of security. You may also need to use the stronger level if you do not know the other person well enough for the common sense questions to be useful.

### **Trading public keys to get stronger security**



The most secure way to trade public keys is to put them on floppy disks and meet in person to trade them, so that you can be absolutely sure that the public key you get does belong to the person who handed it to you. Once you have at least one public key whose ownership you are absolutely sure of, you can use that to help you get public keys through a second way that is only slightly less secure.

This second way involves trusting the owner of that public key to tell you who other public keys belong to, via digitally signed messages. If you are absolutely sure you have the right public key for person A, and you trust person A's judgement, then a digitally signed message from person A stating that person A has made absolutely sure that the included public key belongs to person B may be considered almost as secure as meeting with person B yourself. And as long as the message containing person B's key and statement from person A is digitally signed by person A, it can be distributed through any web page or public database without worrying about further security.



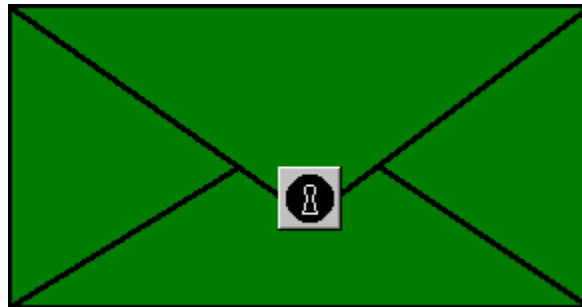
# YOUR ELECTRONIC MAIL SECURITY SOFTWARE

## Locks

Your electronic mail security software provides two kinds of locks for protecting your mail messages against unauthorized reading (eavesdropping) and unauthorized modification (tampering or forgery).



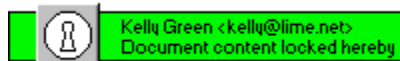
To protect a message against unauthorized reading, put an **envelope lock** on it. Then only an authorized person will be able to open the envelope lock and read the message.



An envelope locked message



To protect a message against unauthorized modification, including forgery, put a **content lock** on it. People who view the message will then be able to see who locked the message and to verify that no later modification has occurred.



A content lock on a message

Both kinds of locks must be used with a **key**.

## Keys



Each person who uses the security software must have their own matched pair of keys, consisting of one black **secret key** and one white **public key**. Secret keys must be carefully protected so that only their owners have access to them. Public keys, however, are meant to be freely distributed and traded, so you should give your public key to everyone you send messages to and get their public keys in return. You can use the **generate-keys** function to make a key pair for yourself.

Secret keys are used for proving identity. They can be used this way while still remaining secret because of the special power of a matched key pair. If a public key is used to put an envelope lock on a message, then only the matching secret key can open that envelope lock. Likewise, if a public key matches a content lock on a message, then that proves that the content lock was created with the matching secret key.

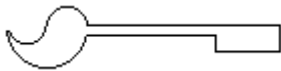
- To protect a message so that only a specific person can read it, use that person's public key to put an envelope lock on it.
- To open an envelope lock on a message that has been sent to you, use your secret key.
- To protect a message so that people can be sure it hasn't been forged, or changed since you protected it, use your secret key to put a content lock on it.
- To verify that no-one has changed a message that has been sent to you since it was content locked, and to verify the identity of the person who content locked it, use that person's public key to check the content lock.

### **Protecting your secret key**



Because your secret key is your proof of identity, you need to carefully protect it against theft or loss. When you create your key pair, your software will ask you to choose a pass phrase (like a password, but longer, since it has multiple words). Your software will not let anyone access your secret key unless they have that pass phrase, so choose something that you will find easy to remember, but that other people won't be able to guess. Then make a backup copy of your key pair and store it in a safe place.

### **Trading public keys to get basic security**

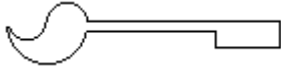


The security of your messages depends on having the right public key for each person. If an attacker can trick you into thinking their public key belongs to someone you send messages to, then the attacker can open envelope locked messages you send to that person, and forge content locked messages from that person to you. When you trade public keys, you need to take precautions to make sure you aren't being tricked.

The simplest way to trade public keys is usually to send them in mail messages or put them up on personal web pages for downloading. The risk is that an attacker could set up a fake web page or forge an email message so that it appears to be from someone you know. For basic security, protect yourself against these kinds of tricks by asking common sense questions. Have you been to this person's web page before, and is it at a web address you know that person uses? Does the message with the key in it sound like that person, and mention things that person would know? Does it come from an email address that you know that person uses? Likewise, when you send your public key to other people, include a note that will help them be sure the message came from you.

This level of security is enough to protect your messages against random eavesdropping and simple forgery, and against attackers who are looking for general vulnerabilities and have no reason to work hard to target your messages in particular. If your messages contain very sensitive or valuable data, or if you have some other reason to think an attacker might want to single you out as a target, then you should consider a stronger level of security. You may also need to use the stronger level if you do not know the other person well enough for the common sense questions to be useful.

### **Trading public keys to get stronger security**



The most secure way to trade public keys is to put them on floppy disks and meet in person to trade them, so that you can be absolutely sure that the public key you get does belong to the person who handed it to you. Once you have at least one public key whose ownership you are absolutely sure of, you can use that to help you get public keys through a second way that is only slightly less secure.

This second way involves trusting the owner of that public key to tell you who other public keys belong to, via content locked messages. If you are absolutely sure you have the right public key for person A, and you trust person A's judgement, then a content locked message from person A stating that person A has made absolutely sure that the included public key belongs to person B may be considered almost as secure as meeting with person B yourself. And as long as the message containing person B's key and statement from person A is content locked by person A, it can be distributed through any web page or public database without worrying about further security.

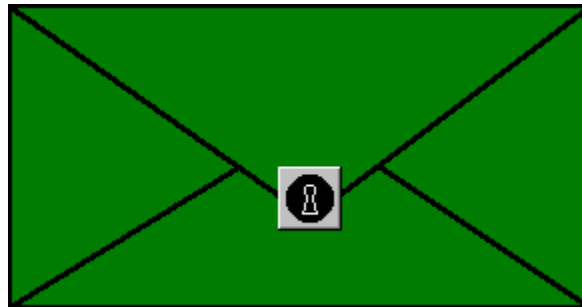
## YOUR ELECTRONIC MAIL SECURITY SOFTWARE

### Locks

Your electronic mail security software provides two kinds of locks for protecting your mail messages against unauthorized reading (eavesdropping) and unauthorized modification (tampering or forgery).



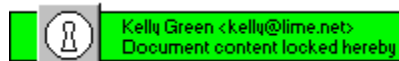
To protect a message against unauthorized reading, put an **envelope lock** on it. Then only an authorized person will be able to open the envelope lock and read the message.



An envelope locked message

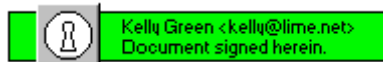


To protect a message against unauthorized modification, including forgery, put a **content lock** on it. People who view the message will then be able to see who locked the message and to verify that no later modification has occurred.



A content lock on a message

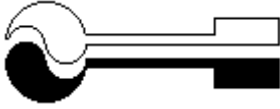
You can also use a special kind of content lock, called a **signature lock**, which provides all the protection of a content lock but also counts as your legal signature when you put it on a message.



A signature lock on a message

Both kinds of locks must be used with a **key**.

## Keys



Each person who uses the security software must have their own matched pair of keys, consisting of one black **secret key** and one white **public key**. Secret keys must be carefully protected so that only their owners have access to them. Public keys, however, are meant to be freely distributed and traded, so you should give your public key to everyone you send messages to and get their public keys in return. You can use the **generate-keys** function to make a key pair for yourself.

Secret keys are used for proving identity. They can be used this way while still remaining secret because of the special power of a matched key pair. If a public key is used to put an envelope lock on a message, then only the matching secret key can open that envelope lock. Likewise, if a public key matches a content lock on a message, then that proves that the content lock was created with the matching secret key.

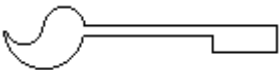
- To protect a message so that only a specific person can read it, use that person's public key to put an envelope lock on it.
- To open an envelope lock on a message that has been sent to you, use your secret key.
- To protect a message so that people can be sure it hasn't been forged, or changed since you protected it, use your secret key to put a content lock on it.
- To verify that no-one has changed a message that has been sent to you since it was content locked, and to verify the identity of the person who content locked it, use that person's public key to check the content lock.

## Protecting your secret key



Because your secret key is your proof of identity, you need to carefully protect it against theft or loss. When you create your key pair, your software will ask you to choose a pass phrase (like a password, but longer, since it has multiple words). Your software will not let anyone access your secret key unless they have that pass phrase, so choose something that you will find easy to remember, but that other people won't be able to guess. Then make a backup copy of your key pair and store it in a safe place.

## Trading public keys to get basic security

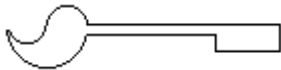


The security of your messages depends on having the right public key for each person. If an attacker can trick you into thinking their public key belongs to someone you send messages to, then the attacker can open envelope locked messages you send to that person, and forge content locked messages from that person to you. When you trade public keys, you need to take precautions to make sure you aren't being tricked.

The simplest way to trade public keys is usually to send them in mail messages or put them up on personal web pages for downloading. The risk is that an attacker could set up a fake web page or forge an email message so that it appears to be from someone you know. For basic security, protect yourself against these kinds of tricks by asking common sense questions. Have you been to this person's web page before, and is it at a web address you know that person uses? Does the message with the key in it sound like that person, and mention things that person would know? Does it come from an email address that you know that person uses? Likewise, when you send your public key to other people, include a note that will help them be sure the message came from you.

This level of security is enough to protect your messages against random eavesdropping and simple forgery, and against attackers who are looking for general vulnerabilities and have no reason to work hard to target your messages in particular. If your messages contain very sensitive or valuable data, or if you have some other reason to think an attacker might want to single you out as a target, then you should consider a stronger level of security. You may also need to use the stronger level if you do not know the other person well enough for the common sense questions to be useful.

### Trading public keys to get stronger security



The most secure way to trade public keys is to put them on floppy disks and meet in person to trade them, so that you can be absolutely sure that the public key you get does belong to the person who handed it to you. Once you have at least one public key whose ownership you are absolutely sure of, you can use that to help you get public keys through a second way that is only slightly less secure.

This second way involves trusting the owner of that public key to tell you who other public keys belong to, via another special kind of content lock called a **certification lock**. If you are absolutely sure you have the right public key for person A, and you trust person A to make sure that they themselves have the right public key for person B, then getting a public key for person B that has been certification locked by person A may be considered almost as secure as meeting with person B yourself. As long as person B's public key is certification locked by person A, it can be distributed through any web page or public database without worrying about further security.



A certification lock on a public key

## C.2 Questions and collated results

### Participant demographics (8 participants)

#### PGP metaphors

| P#  | age | education            | expertise                         |
|-----|-----|----------------------|-----------------------------------|
| P66 | 52  | M.S. degree          | Real estate                       |
| P61 | 22  | some college         | drama (acting)                    |
| P60 | 41  | some college         | administration                    |
| P56 | 19  | some college         | engineering                       |
| P55 | 20  | some college         | psychology                        |
| P51 | 29  | some graduate school | business                          |
| P52 | 24  | M.S. degree          | physics                           |
| P46 | 24  | B.S. degree          | electrical & computer engineering |

#### Basic Lime metaphors

| P#  | age | education    | expertise                           |
|-----|-----|--------------|-------------------------------------|
| P67 | 52  | J.D. degree  | lawyer                              |
| P54 | 19  | some college | information systems                 |
| P68 | 23  | some college | mechanical engineering              |
| P43 | 22  | B.S. degree  | civil and environmental engineering |
| P44 | 21  | some college | civil engineering                   |
| P50 | 18  | some college | managerial economics                |
| P59 | 19  | some college | political science                   |
| P57 | 21  | some college | business                            |
| P63 | 26  | M.A. degree  | foreign language education          |

#### Extended Lime metaphors

| P#  | age | education            | expertise                                |
|-----|-----|----------------------|--|
| P45 | 27  | M.S. degree          | environmental science/engineering/policy |
| P47 | 25  | some graduate school | systems engineering                      |
| P48 | 21  | some college         | information and decision systems         |
| P49 | 21  | some college         | history and education                    |
| P53 | 23  | B.A.                 | chemistry                                |
| P58 | 20  | some college         | psychology                               |
| P64 | 24  | M.S. degree          | materials science and engineering        |
| P65 | 21  | some college         | natural sciences                         |
| P62 | 19  | some college         | business                                 |

Time spent on each portion of test session (minutes), not including debriefing questionnaire

PGP metaphors

| P#  | reading prior to requesting first question | Question 1 | Question 2 | Question 3 | Question 4 | Question 5 | Question 6 | total |
|-----|--|------------|------------|------------|------------|------------|------------|-------|
| P66 | 8  | 12         | 3          | 15         | 11         | 7          | 1          | 57    |
| P61 | 5  | 7          | 6          | 4          | 2          | 4          | 2          | 30    |
| P60 | 5  | 1          | 6          | 4          | 5          | 4          | 3          | 28    |
| P56 | 6  | 6          | 4          | 4          | 2          | 6          | 2          | 30    |
| P55 | 7  | 10         | 10         | 5          | 6          | 6          | 6          | 50    |
| P51 | 10   | 2          | 3          | 12         | 12         | 11         | 12         | 62    |
| P52 | 7  | 6          | 5          | 5          | 4          | 6          | 6          | 39    |
| P46 | 10   | 4          | 6          | 6          | 6          | 4          | 4          | 42    |

Basic Lime metaphors

| P#  | reading prior to requesting first question | Question 1 | Question 2 | Question 3 | Question 4 | Question 5 | Question 6 | total |
|-----|--|------------|------------|------------|------------|------------|------------|-------|
| P67 | 8  | 2          | 6          | 4          | 5          | 1          | 8          | 34    |
| P54 | 6  | 2          | 4          | 4          | 3          | 8          | 3          | 30    |
| P68 | 7  | 12         | 7          | 6          | 5          | 4          | 10         | 51    |
| P43 | 5  | 1          | 7          | 5          | 5          | 3          | 3          | 24    |
| P44 | 7  | 4          | 9          | 7          | 5          | 6          | 6          | 44    |
| P50 | 4  | 7          | 5          | 3          | 3          | 5          | 3          | 30    |
| P59 | 4  | 1          | 5          | 3          | 4          | 3          | 4          | 24    |
| P57 | 7  | 1          | 4          | 3          | 2          | 2          | 3          | 22    |
| P63 | 9  | 1          | 5          | 4          | 2          | 4          | 2          | 27    |

Extended Lime metaphors

| P#  | reading prior to requesting first question | Question 1 | Question 2 | Question 3 | Question 4 | Question 5 | Question 6 | total |
|-----|--|------------|------------|------------|------------|------------|------------|-------|
| P45 | 9  | 1          | 4          | 3          | 4          | 4          | 3          | 28    |
| P47 | 5  | 7          | 3          | 5          | 3          | 4          | 2          | 29    |
| P48 | 5  | 2          | 2          | 4          | 4          | 3          | 4          | 24    |
| P49 | 8  | 3          | 3          | 8          | 4          | 6          | 4          | 36    |
| P53 | 8  | 1          | 1          | 5          | 4          | 3          | 4          | 26    |
| P58 | 6  | 2          | 4          | 7          | 3          | 3          | 4          | 29    |



|     |   |    |   |   |   |   |   |    |
|-----|---|----|---|---|---|---|---|----|
| P64 | 7 | <1 | 6 | 5 | 5 | 5 | 4 | 32 |
| P65 | 8 | 4  | 6 | 5 | 4 | 5 | 4 | 36 |
| P62 | 4 | 2  | 3 | 3 | 2 | 4 | 2 | 20 |

## Question 1

You need to send an email message to your housemates reminding them to buy lightbulbs, since your household is nearly out.

(1) Would you, in real life, think it was worth putting in some extra time to make this message secure, rather than simply sending it in a regular email? If yes, how much extra time (in seconds, minutes, hours, or days) would you think it was worth?

### PGP metaphors

|     |   |
|-----|---|
| P66 | yes, 2 minutes or whatever it would take to secure their privacy on matters of where they live or any information that someone might use to deduce their life style   |
| P61 | No I wouldn't secure it. If there was a situation where I didn't want someone to know who I lived with, I might but I wouldn't want to spend more than 30 seconds on it.  |
| P60 | No I would just use regular email   |
| P56 | Yes if it takes less than 30 seconds  |
| P55 | Yes, I would want to make sure that everything I send, no matter how unimportant it is, is only read by who it is written to. I would spend more time protecting more important messages, but one about lightbulbs I'd spend an extra 5 minutes on. |
| P51 | No  |
| P52 | Yes, 5 seconds or less.   |
| P46 | No, I don't think it's worth to put in extra time for this message  |

### Basic Lime metaphors

|     |   |
|-----|---|
| P67 | No.   |
| P54 | No I don't feel it is worth putting in extra time to make this message secure.  |
| P68 | Well since I want my housemates to be the only part to receive the email, I do feel one should put a little time into securing the message. How much time? ... Probably an hour or two. |
| P43 | No  |
| P44 | Highly doubt it is worth the time.  |
| P50 | Yes. No more than 10 extra seconds.   |
| P59 | No. Regular email would be fine.  |
| P57 | NO  |
| P63 | No  |

## Extended Lime metaphors

|     |   |
|-----|---|
| P45 | No  |
| P47 | in the case of lightbulbs, I would use security measures if it no more than an extra 30-45 seconds (good habits)                  |
| P48 | regular e-mail  |
| P49 | I don't think any extra security is necessary. The worst that could happen is someone else knows I need bulbs.                    |
| P53 | no  |
| P58 | No, I wouldn't bother securing this message. I wouldn't care if someone knew about our lack of lightbulbs.                        |
| P64 | no  |
| P65 | For something as basic and general as buying lightbulbs, I wouldn't go to any extra trouble to secure it. A few seconds, at most. |
| P62 | No b/c if another person reads this email its not like they would be lightbulbs for you anyways.                                  |

(2) If you answered “yes” to question 1, then can you tell, from the software description you were given, which keys and which functions you and your housemates would each need to use? If yes, please list them.

## PGP metaphors

|     |   |
|-----|---|
| P66 | Send by public key - secret key to sign. Read by secret key - public key to verify.   |
| P61 | I would need their public keys to encrypt the message and they would use their secret keys to decrypt it.   |
| P60 | <blank>   |
| P56 | you: encrypt, housemates public key. housemates: decrypt, private key.  |
| P55 | I would have to get public keys from both my housemates I think using their public key to send a message encrypts the message and digitally signs it for me so that only they can decrypt and verify the signature. |
| P51 | <blank>   |
| P52 | (1) use my housemates public key to encrypt message (2) sign it using my secret key   |
| P46 | <blank>   |

## Basic Lime metaphors

|     |  |
|-----|--|
| P67 | <blank>  |
| P54 | <blank>  |
| P68 | We would all have our secret keys in which we would know each other's identity, also with our public keys we would be able to know from who the particular email was sent from. In terms of locks, I would provide the message with both types of locks, i.e., envelope lock & content lock. |
| P43 | <blank>  |
| P44 | If I did send this message, I would simply use an envelope lock. This would require me having the public keys to my housemates and all of them   |

|     |   |
|-----|---|
|     | having my key.  |
| P50 | Yes. I can put a content lock on the email. I can use a public key. |
| P59 | <blank>   |
| P57 | <blank>   |
| P63 | <blank>   |

### Extended Lime metaphors

|     |  |
|-----|--|
| P45 | <blank>  |
| P47 | I would need my housemates' public key, my housemate would need his private key. |
| P48 | <blank>  |
| P49 | <blank>  |
| P53 | <blank>  |
| P58 | <blank>  |
| P64 | <blank>  |
| P65 | I would secure it with my secret key so they can open it with my public key.     |
| P62 | <blank>  |

(3) If you answered “yes” to question 1, then can you tell, from the software description you were given, what steps you and your housemates would each need to take to get those keys? If yes, please list them.

### PGP metaphors

|     |   |
|-----|---|
| P66 | Send an email message a person you know well. "If the shoe fits well, buy it." If the shoes are a gift include a card so they know who to thank.  |
| P61 | Since I would see them all the time, we would exchange public keys in person. We could send them in a mail message as it would be reasonably easy to trust that it's them. We would need to think up pass phrases for our own secret keys.  |
| P60 | <blank>   |
| P56 | You could send them to each other in a mail message or put them up on personal web page for downloading.  |
| P55 | To receive an email from me, I don't think my housemates need to "get" any keys from me. They would need to create a set of keys, then trade the public keys w/me. They could save them onto disks + trade them with me, or only one housemate could do that and since I trust that person, they could email me my other housemates public key. |
| P51 | <blank>   |
| P52 | (1) I will ask my housemate for his public key meeting in person by a regular email (2) will tell him my public key either by meeting him or by regular email   |
| P46 | <blank>   |

## Basic Lime metaphors

|     |  |
|-----|--|
| P67 | <blank>  |
| P54 | <blank>  |
| P68 | From the software description, my housemates and I would need to provide each other with our corresponding public keys. According to the description the best way to achieve this would be to physically hand one of my housemates say a disk with my public key information since I have complete confidence in my housemates, there would be no problem in obtaining the other public keys. This would be the safest way to achieve this task. |
| P43 | <blank>  |
| P44 | Basically, we could trade keys via disks or download. Trust factor would be very high, as we all live together.  |
| P50 | I would have to use the generate-keys function. For my secret key, I have to come up with a pass phrase, something like a password, but with many words instead of one.  |
| P59 | <blank>  |
| P57 | <blank>  |
| P63 | <blank>  |

## Extended Lime metaphors

|     |  |
|-----|--|
| P45 | <blank>  |
| P47 | 1. Each generate a key pair. 2. Exchange public keys. 3. I would need to enter his/her public key as part of the message. 4. He would need to use his private key to obtain the message.       |
| P48 | <blank>  |
| P49 | <blank>  |
| P53 | <blank>  |
| P58 | <blank>  |
| P64 | <blank>  |
| P65 | They wouldn't necessarily need their own secret key, but would need a copy of my public key. I would have to create a matched set of keys and give them a copy via floppy disk or downloading. |
| P62 | <blank>  |

(4) Are there any comments you would like to make?

## PGP metaphors

|     |   |
|-----|---|
| P66 | There seems to be too much confusion in the clips involved. It reminds me of a WWII information pass off.   |
| P61 | I had questions...what exactly is the exchange of a public key? Would it be a password? A code or i.d. #? Also, after verifying the digital signature, does that then go away to show that the message has been looked at or tampered with? As a computer illiterate, I don't have the resources to make assumptions to answer these questions. |

|     |   |
|-----|---|
| P60 | <blank>   |
| P56 | <blank>   |
| P55 | I don't understand whether or not someone with the same public key as I have could read a message that I send the person with a secret key that matches it. |
| P51 | <blank>   |
| P52 | These encryption/decryptions are not so important for buying "lightbulbs" or like!!   |
| P46 | <blank>   |

### Basic Lime metaphors

|     |   |
|-----|---|
| P67 | The information is trivial and of value to no one.  |
| P54 | <blank>   |
| P68 | <blank>   |
| P43 | <blank>   |
| P44 | Seems that such difficult encryption would be better employed elsewhere.  |
| P50 | It takes reading through twice to understand the description. Add more pictures to follow along with the description. |
| P59 | <blank>   |
| P57 | <blank>   |
| P63 | <blank>   |

### Extended Lime metaphors

|     |  |
|-----|--|
| P45 | <blank>  |
| P47 | If this process were automated, so that I would only need to enter a key once, I would use this method frequently. |
| P48 | <blank>  |
| P49 | <blank>  |
| P53 | <blank>  |
| P58 | No.  |
| P64 | <blank>  |
| P65 | No   |
| P62 | Sending lightbulbs is not private.   |

## Question 2

You want to send an email message to an old friend from school to bring them up to date on some stressful developments in your personal life. You haven't talked to them in a year or so, since they now live on the other side of the country, but you have their email address at the company they currently work for.

(5) Would you, in real life, think it was worth putting in some extra time to make this

message secure, rather than simply sending it in a regular email? If yes, how much extra time (in seconds, minutes, hours, or days) would you think it was worth?

### PGP metaphors

|     |   |
|-----|---|
| P66 | no  |
| P61 | If the stressful developments were personal enough, then yes. I would give it a few minutes of my time.                         |
| P60 | I would use the secure method but spend no more than 10 minutes trying to implement it.   |
| P56 | Yes if it took less than 30 seconds   |
| P55 | If I haven't talked to them in a year, I would be willing to put off mailing the message a few days to make sure it was secure. |
| P51 | No  |
| P52 | 1-1.5 minutes   |
| P46 | Yes, up to 10 min.  |

### Basic Lime metaphors

|     |   |
|-----|---|
| P67 | Possibly yes. I would expect to spend perhaps 30 minutes initially learning the system. Gradually I would want each application to take less than 30 seconds.                               |
| P54 | Yes, an extra 3-5 min.  |
| P68 | Yes. This situation here is much more personal and would need to be received by your friend and only your friend. One would need to provide a lot more time, say one to four days at least. |
| P43 | Yes, a few (<5) minutes   |
| P44 | Yes, depends on how personal the developments were. I'd spend 5-10 minutes.   |
| P50 | I would say extra minutes.  |
| P59 | Yes, I would want to make this message safe. I would take 5 extra minutes to make it secure.  |
| P57 | Yes, 30 seconds to 1 minute   |
| P63 | Yes, in 30 seconds  |

### Extended Lime metaphors

|     |   |
|-----|---|
| P45 | Yes, 5 min.   |
| P47 | no. many companies dislike using secure methods of email -- I'd email him/her and ask for address for private/social email. |
| P48 | 15 min  |
| P49 | I don't think extra security is needed. The worst case is someone I don't know finds out some personal info about me.       |
| P53 | no  |
| P58 | I suppose if the email was very personal then I would take maybe 2 minutes to secure it.                                    |
| P64 | yes. 2 minutes  |
| P65 | Since I don't see this person everyday, I might be willing to spend a few   |

|     |                                 |
|-----|---------------------------------|
|     | minutes securing it.            |
| P62 | Yes, about an extra 10 minutes. |

(6) If you answered “yes” to question 5, then can you tell, from the software description you were given, which keys and which functions you and your friend would each need to use? If yes, please list them.

### PGP metaphors

|     |   |
|-----|---|
| P66 | <blank>   |
| P61 | I would need her public key and she would need her private key to decrypt. I would encrypt it.  |
| P60 | secret key for me public key for person I am sending to to encrypt message  |
| P56 | you: encrypt, friend's public key. friend: decrypt, secret key.   |
| P55 | I need to get her public key, and then send her a message using that and only she has the secret key to match, so only she could read it.   |
| P51 | <blank>   |
| P52 | (1) I need to use my friend's public key to encrypt the message. (2) I need to use my secret key to sign it. (3) My friend needs his secret key to decrypt. (4) My friend need my public key to verify signature. |
| P46 | Functions I will be using are encrypt, digital signature and generate keys. The keys are secret key. My friend will be using decrypt and verify-signature fn, and public key                                      |

### Basic Lime metaphors

|     |   |
|-----|---|
| P67 | envelope lock. He/she would need to give me his/her public key. He/she would use his/her private key to open it.  |
| P54 | I would have to use either his public and he would use his private if I found a way to get his public or he could use my public and I would use my private.   |
| P68 | Again, since you would like to have your mail (message) received by the party that you intended to receive it, one would need to use both locks, envelope and content, on the mail message. For this particular question, the importance and significance of the message is not relevant. All mail should be considered secret. |
| P43 | Envelope lock - use their public key to lock message, they will need their secret key to open message. Content lock - use my secret key to lock message, they will use my public key to open message.   |
| P44 | We would each need our own secret keys, of course, and the public keys of each other. Both types of locks could be employed but primarily the envelope lock.  |
| P50 | I would need both kinds of locks and a secret key.  |
| P59 | <u>Yes</u> , envelope + content lock  |
| P57 | We would use public and private keys, and there would be envelope and content locks.  |
| P63 | public key  |

## Extended Lime metaphors

|     |  |
|-----|--|
| P45 | Envelope lock with a public key & secret key   |
| P47 | <blank>  |
| P48 | envelope lock  |
| P49 | <blank>  |
| P53 | <blank>  |
| P58 | - envelope lock to secure message. - I would need their public key to lock it. - they would need their secret key to read it.  |
| P64 | yes - I would use my secret key to lock the content of the message, so that my friend was sure that it hadn't been tampered with + I would use their public key to envelope lock the message so only they could read it. They would have to use their secret key to open it. |
| P65 | I would put an envelope lock and a content lock on it. I would secure it with my secret lock so that, assuming he can obtain one, he can open it with a copy of my public key.   |
| P62 | Signature lock and send them a secret key.   |

(7) If you answered “yes” to question 5, then can you tell, from the software description you were given, what steps you and your friend would each need to take to get those keys at an appropriate level of security? If yes, please list them.

## PGP metaphors

|     |  |
|-----|--|
| P66 | <blank>  |
| P61 | First I would email her and asked questions in order to verify that it's her. Then when I knew and she knew it was me, we would exchange public keys through email w/out identifying them within the email in case of eavesdropping.   |
| P60 | secret key for me public key on a floppy and meeting the person to make sure   |
| P56 | Because it is not that important, basic security will suffice. Therefore posting them on a web page for downloading would be an appropriate level.   |
| P55 | <del>I guess since I can't see her in person, I would first send an email asking her a question that only she would know. Once I verified her email address is correct, I would ask her to call me. Then tell her a website where she could download my public key. No, then</del> |
| P51 | <blank>  |
| P52 | (1) I will exchange some general emails talking about old school days to be comfortable that he is the genuine person and is interested still. (2) will send him the public key on regular email.  |
| P46 | I will send a email to him and get his phone no. and then will tell him my public key to him and will ask his public key.  |

## Basic Lime metaphors

|     |  |
|-----|--|
| P67 | Exchange in person if possible. Exchange through downloading if necessary would be ok, most likely, since need is only for basic security. |
| P54 | I could either e-mail him my public key and he could email me his public or if I had a webpage set up he could download it from there.     |



|     |  |
|-----|--|
| P68 | In this case, it would be expensive (flight) to personally hand someone a disk with my public key and what not. The approach which needs to be taken would be to simply send the email with both types of locks to the friend.           |
| P43 | We could exchange our public keys using e-mail. In the email message we should start a conversation that would verify us. This would involve something that only we might know such as a knickname.                                      |
| P44 | Technically, we could mail disks to each other. We could also trade via unsecured e-mail, using verification phrases first to make sure these friends are who they say they are.   |
| P50 | We have to generate keys. But first, I would write a regular email and ask if they cared to hear my problems. After keys are generated, we exchange public keys.   |
| P59 | I would need my friend's public key for envelope lock. He would need his secret key to open the envelope lock. I would have to use my secret key for the content lock. He would use my public key to check the content lock.             |
| P57 | The secret keys are only ours, but we must trade public keys. I would trade the public key through an email message, with something to let them know it was from me, that is something that I do differently than others, and vice-versa |
| P63 | I will send the public key in mail message.  |

### Extended Lime metaphors

|     |  |
|-----|--|
| P45 | Not really   |
| P47 | <blank>  |
| P48 | - would need to trade public keys - friend must have secret key  |
| P49 | <blank>  |
| P53 | <blank>  |
| P58 | - she could either email me her public key, put it on a webpage, or give it to me in person.                           |
| P64 | yes - my friend would've had to generated their own set of keys + then would have had to email their public key to me. |
| P65 | I would have to create a matched key set and, since they are far apart, put a copy on a website for downloading.       |
| P62 | Secret key along with a certification lock.  |

(8) Are there any comments you would like to make?

### PGP metaphors

|     |   |
|-----|---|
| P66 | Any matters that would cause stress in my life would be of public record, and I would not use a business ? mail to disclose anything of a private matter. Privacy laws are not universal.   |
| P61 | This seems complicated! Really with this amount of beforehand preparation, I would have to be sending something really secret to want to do this. I don't really care that much if a stranger finds out I can't pay my cable bill or I'm failing history. |
| P60 | It seems too much like a James Bond movie   |

|     |  |
|-----|--|
| P56 | <blank>  |
| P55 | Well, if email is not secure, how can I stop people from getting my public key that I send out if I can't see them in person.    |
| P51 | <blank>  |
| P52 | I will be rather careful discussing issues in personal life at the risk of eavesdropping.  |
| P46 | After answering (7), I think, I will rather talk to him on my personal life on phone, rather than doing what I mentioned in (7). |

### Basic Lime metaphors

|     |  |
|-----|--|
| P67 | <blank>  |
| P54 | <blank>  |
| P68 | <blank>  |
| P43 | <blank>  |
| P44 | the content lock seems useless. why would you want to bother content locking something that people could still read when you could simply envelope lock it and nobody could read it without a key. And as for the forgeries, unless you are sure of key sources, anyone could steal a key and envelope lock a message that's a forgery. Actually scratch all that, I just re-read that you need your secret key to set up a content lock. That would be harder to forge. |
| P50 | Nope   |
| P59 | <blank>  |
| P57 | <blank>  |
| P63 | <blank>  |

### Extended Lime metaphors

|     |  |
|-----|--|
| P45 | <blank>  |
| P47 | It seems that if you were able to obtain (reliably) a public key from a friend, you could also get a private email.                                    |
| P48 | <blank>  |
| P49 | <blank>  |
| P53 | <blank>  |
| P58 | No.  |
| P64 | <blank>  |
| P65 | I wouldn't know if my friend has a copy of the key without having read the email, which needs a key to be opened. Knowing this, I might not secure it. |
| P62 | Seems like a few options are available.  |

### Question 3

Your Internet Service Provider (ISP) sometimes needs to send you email messages telling you to make certain changes in your software. If a hacker was able to forge one of those

messages to you, so that you believed it had come from your ISP, then he or she might be able to break into your computer.

(9) Would you, in real life, think it was worth putting in some extra time to make sure this message was secure, rather than simply trusting a regular email? If yes, how much extra time (in seconds, minutes, hours, or days) would you think it was worth?

#### PGP metaphors

|     |   |
|-----|---|
| P66 | yes 2 days if the material sent were sensitive                          |
| P61 | Yes. Fifteen minutes.   |
| P60 | 15-20 minutes   |
| P56 | Yes. 2 minutes.   |
| P55 | Yes, I'd spend a day or two verifying that the message was from my ISP. |
| P51 | Yes, minutes  |
| P52 | 2-10 minutes  |
| P46 | Yes, couple of days.  |

#### Basic Lime metaphors

|     |   |
|-----|---|
| P67 | Yes. One hour initially; gradually working down to 30 seconds or less per use.  |
| P54 | Yes, take a couple of hours to meet in person.  |
| P68 | Definitely. Computers are expensive and need to be protected. I would suggest putting in about 21 days. Much more than the previous two situations.             |
| P43 | Yes, 10 minutes.  |
| P44 | Yes, I would definitely be worth it. Time, I would say as long as necessary, but in reality, anything over say 20-30 minutes will get old if you do this a lot. |
| P50 | It would be worth extra hours. I would hate for someone to be able to break into my computer.   |
| P59 | Yes. I would put an hour into protection if there were possibility of someone's breaking into my computer.  |
| P57 | Yes, 1 minute - 2 minutes   |
| P63 | Yes, 1 minute.  |

#### Extended Lime metaphors

|     |   |
|-----|---|
| P45 | Yes, 30 min.  |
| P47 | Yes. I would spend at most 10 extra minutes.  |
| P48 | 1.5 hours   |
| P49 | Yes, perhaps a few minutes at most.   |
| P53 | yes, 1-2 min.   |
| P58 | I guess I would spend maybe 5 minutes to secure the message. I tend to think that no one would want to break into my computer though. |
| P64 | yes -10 minutes   |

P65 I would spend a longer time securing this. Up to several hours.

P62 Yes, all the time so it does not happen.

(10) If you answered “yes” to question 9, then can you tell, from the software description you were given, which keys and which functions you and your ISP would each need to use? If yes, please list them.

### PGP metaphors

P66 (1) the person public key to encrypt message sent, secret key to sign message. (2) Secret key to decrypt, use public to verify signature.

P61 If I had already given my public key to my ISP or if I had theirs? then I would expect messages to be digitally signed and poss. also encrypted. If we had not already exchanged in some secure way, then I don't know how I would verify the identity of the sender.

P60 Digital signature function to make sure from a legitimate source

P56 you: decrypt, secret key, verify-signature. ISP: encrypt, your public key, digital signature.

P55 They (my ISP) would need to get my public key in order to send email that only I could read. I would need their public key to verify the signature.

P51 verify-signature, public key

P52 I would need (1) ISP's public key to encrypt my message to them and to verify the signature. ISP needs (1) my public code to encrypt the message to me (2) a secret code to sign it.

P46 same as answer 6. Instead of a friend, there will be someone from ISP.

### Basic Lime metaphors

P67 Both envelope lock and content lock needed. You would need each other's public keys.

P54 Whoever was sending the message would use their private key to ensure a content lock. And, the receiver of the mail would use the other person's private key.

P68 One would need to have the paired keys, secret and public. Since this situation deals with breaking into a system one would need to have the mail message with both envelope and content locks.

P43 Content lock - they use their secret key to lock - use their public key to be sure message hasn't changed.

P44 My ISP would need their secret key to content lock and my public key to message lock. I would need my secret key to unlock the message and the ISP's public key to verify that it hasn't been changed.

P50 Locks: 1) content lock 2) envelope lock. Keys: 1) public key for everyone with the same ISP 2) secret key for your personal email instructions.

P59 ISP would need to put a content lock on it so it couldn't be forged. Eavesdropping is unimportant in this case.

P57 Public/private, content, envelope

P63 An envelope lock & secret key

### Extended Lime metaphors

|     |  |
|-----|--|
| P45 | Content lock with public key & secret key  |
| P47 | I would need the sysadmins public key. The email would require a content lock.   |
| P48 | envelope, content, and signature lock  |
| P49 | I would want to use a signature lock so that if there was a problem they would be liable. This would be used w/ <del>my secret key</del> their public secret key.                                |
| P53 | yes. ISP: secret key - content lock. me: ISP public key - content lock.  |
| P58 | - the ISP would need to put a content lock on the message. - I would need the ISP's public key to verify that there were no changes.   |
| P64 | yes - I would need a public key from the ISP in order to check that their message had not been tampered with. The ISP would need my public key to lock the message so that only I could read it. |
| P65 | I would want the envelope and content locks put on. They would send it using my public key and I would open it with my secret key.   |
| P62 | content lock   |

(11) If you answered “yes” to question 9, then can you tell, from the software description you were given, what steps you and your ISP would each need to take in order to get those keys at an appropriate level of security? If yes, please list them.

### PGP metaphors

|     |  |
|-----|--|
| P66 | <blank>  |
| P61 | Not really. Exchange at the very beginning when I am signing up and know for sure I am talking to my ISP I guess.  |
| P60 | secret key, public key   |
| P56 | At this level of security I would trade public keys on floppy disks.   |
| P55 | maybe when they set up the internet in my house they would give me a disk with their public key on it and then I could email them my public key  |
| P51 | Use the public key to encrypt the message then use the secret key to decrypt the message. Use the secret key to digital signature the message. Finally use the public key to verify signature. |
| P52 | I will talk to an authorized representative of the ISP in person or will tell them my public key while signing out for their service in the beginning.   |
| P46 | Instead of conversation on phone, I will like to meet the person personally will like to verify the security of my public key.   |

### Basic Lime metaphors

|     |  |
|-----|--|
| P67 | Exchange floppy disks in person containing public keys.  |
| P54 | I believe the only way to be sure would be to put the keys on disk and trade them in person.                                   |
| P68 | Since one might not know very well (say on a personal level) ones ISP, the safest step would be to actually meet with the ISP. |

|     |   |
|-----|---|
| P43 | Their public key could be sent to you in an ordinary message.   |
| P44 | Hopefully, my ISP would mail me a disk with their public key, so that I could send a secure message to them using the message lock containing my key. I could also download their key from a secure website, requiring passwords to log into. |
| P50 | 1) Both my ISP and I have to generate keys. 2) My ISP and I have to trade public and secret keys, via disk exchange.  |
| P59 | ISP would use their secret key for a content lock. I would need to get their public key to check the content lock.  |
| P57 | We might have to meet to exchange the public key, since I don't know anything about the ISP, maybe we could exchange at the time I se-up my account with my ISP.  |
| P63 | make up a pass phrase   |

### Extended Lime metaphors

|     |  |
|-----|--|
| P45 | Not really   |
| P47 | 1. public & private keys for the ISP sysadmin are generated. 2. public key is mailed on a CD or some other read-only media. 3. I would use the key to verify identity.                                 |
| P48 | - both have a secret key - meet to exchange public key   |
| P49 | ISP would need to signature lock the message w/their secret key and opened and verified w/my public key.   |
| P53 | no   |
| P58 | - the ISP needs their secret key to put a content lock on it. - I would need the ISP's public key. They could email it to me, I could download it from a website, or they could give it to me on disk. |
| P64 | for the appropriate level of security our public keys should be traded by floppy in person.  |
| P65 | I would have to create the keys and they would download my public key from an agreed upon website.   |
| P62 | secret key, to prove identity that it is not a hacker  |

(12) Are there any comments you would like to make?

### PGP metaphors

|     |  |
|-----|--|
| P66 | The wording ran to confusing when reading I found myself constantly verifying my terms.  |
| P61 | <blank>  |
| P60 | <blank>  |
| P56 | <blank>  |
| P55 | Can I create as many sets of keys as I want? I could email someone my public key, then have them email me their public key, then I could send them a new public key now that I know only they can read it? |
| P51 | <blank>  |
| P52 | <blank>  |

P46 I think this software could be useful in the above case, if the ISP is an reputed company, so that a person from that company can be trusted.

### Basic Lime metaphors

P67 <blank>

P54 <blank>

P68 <blank>

P43 <blank>

P44 Yeah, this is becoming more clear after 2 examples. Could probably use this now with very minimal screw-ups.

P50 Nope

P59 <blank>

P57 <arrow to above>

P63 <blank>

### Extended Lime metaphors

P45 <blank>

P47 this type of security would have to be arranged when you set up an account w/an ISP.

P48 <blank>

P49 <blank>

P53 <blank>

P58 No.

P64 public keys in this case would be difficult to trade securely over email or the web because of the impersonable nature of the ISP-me relationship.

P65 <blank>

P62 good use for this system is this example

## Question 4

You are involved in a fairly high-stakes lawsuit, and you need to communicate regularly with your lawyer about it by email. You have told your lawyer things about the circumstances of the suit that you expect to be covered by attorney-client privilege, and you need to discuss those things in your email messages.

(13) Would you, in real life, think it was worth putting in some extra time to make these messages secure, rather than simply relying on regular email? If yes, how much extra time (in seconds, minutes, hours, or days) would you think it was worth?

### PGP metaphors

P66 <blank>

P61 yes - 30 minutes

|     |   |
|-----|---|
| P60 | yes 30 minutes  |
| P56 | Yes. 5 min.   |
| P55 | I would spend as much time as I could. I'd spend a few days if I didn't need to get a response immediately. |
| P51 | Yes, minutes.   |
| P52 | 10-15 minutes   |
| P46 | Yes, maybe up to a month if required.   |

### Basic Lime metaphors

|     |   |
|-----|---|
| P67 | Yes. However much time it took to learn how to use the system and however much time it took to use it -- assuming that it would not take an excessive amount of time each use (say 120 seconds) |
| P54 | Yes, as much time as needed.  |
| P68 | Yes. Again since the information being provided via email could dictate one's personal life and future I would say 21 days again.   |
| P43 | Yes, 10 minutes   |
| P44 | Same as the last one, as much time as necessary. If it seriously took longer than 20-30 minutes, I'd meet him somewhere.  |
| P50 | Days  |
| P59 | Yes. I would put a few hours in for security.   |
| P57 | Yes, possibly a few days  |
| P63 | Yes, in 30 seconds  |

### Extended Lime metaphors

|     |  |
|-----|--|
| P45 | Yes, 1 hr.   |
| P47 | Yes. As much time as it takes.   |
| P48 | 2 hrs  |
| P49 | Yes. Upwards to around a 1/2 hour.   |
| P53 | yes, 5 min   |
| P58 | Yes. I might take 10 minutes to secure the mail. Otherwise I would just use the phone. |
| P64 | yes - 10   |
| P65 | Yes, very important that it is secure. As long as it takes. Days, if necessary.        |
| P62 | yes, all the time necessary  |

(14) If you answered "yes" to question 13, then can you tell, from the software description you were given, which keys and which functions you and your lawyer would each need to use? If yes, please list them.

### PGP metaphors

|     |  |
|-----|--|
| P66 | Trade public keys of floppy disk. Send on public key P, sign with secret |
|-----|--|



|     |   |
|-----|---|
|     | S. Read with secret S using pass phrase to access, public to verify P using pass key to verify it.  |
| P61 | each other's public keys, our own secret keys. Encryption + digital signature.  |
| P60 | trading keys, secret key, public key put on floppy and meet in person   |
| P56 | You: encrypt, lawyer's public key, digital signature. Lawyer: decrypt, secret key, verify-signature.  |
| P55 | My lawyer needs my public key to email me securely and I'd need his to email him. I'd also need his to verify his signature and vice versa.   |
| P51 | encrypt/decrypt and digital, verify signature, and secret and public keys   |
| P52 | I need (1) My lawyer's public key to encrypt my message to him. (2) My secret code to sign my message. My lawyer needs (1) his secret code to decrypt my message. (2) My public key to verify that the message has not been tampered. |
| P46 | same as in 6 or 10  |

### Basic Lime metaphors

|     |  |
|-----|--|
| P67 | Use both envelope and content lock. You and he/she would need to exchange public keys on disk.   |
| P54 | The person sending the mail could use their private key while the person receiving the mail would use the senders public key.  |
| P68 | I suggest having both secret and public keys used. Again having both types of locks (envelope and content on it).  |
| P43 | Both content and envelope lock. Content lock - they use their secret key to lock, you use their public key to verify. Envelope lock - they use your public key to lock, you use your secret key to open. |
| P44 | We need our own secret keys and each other's public keys to use both the envelope lock and the content lock.   |
| P50 | We would need public + secret keys and envelope + content locks.   |
| P59 | envelope + content locks. We would need our own public keys + private keys.  |
| P57 | Public/private, content/envelope   |
| P63 | envelope lock  |

### Extended Lime metaphors

|     |  |
|-----|--|
| P45 | Envelope lock & content lock. With their public & secret keys.   |
| P47 | we would each need a unique key pair. Both signature and content locks would be used.  |
| P48 | need secret key & public key, would use envelope, content & signature lock   |
| P49 | I would use a signature lock in conjunction w/our traded public keys on floppy disks.  |
| P53 | yes. me: lawyer's public key - envelope lock. lawyer: secret key to open envelope lock.  |
| P58 | - The lawyer would need my public key to envelope lock it. - I would need my secret key to read it.  |
| P64 | yes - messages would need to be both enveloped + content locked so we would both need our own set of secret/public keys as well as each other's public |

|     |   |
|-----|---|
|     | keys  |
| P65 | I would use envelope, content, and signature locks. I would send an email with my secret key, and my lawyer could open it with my public key. |
| P62 | envelope lock so only lawyer can read   |

(15) If you answered “yes” to question 13, then can you tell, from the software description you were given, what steps you and your lawyer would each need to take to get those keys at an appropriate level of security? If yes, please list them.

### PGP metaphors

|     |  |
|-----|--|
| P66 | <blank>  |
| P61 | Exchange disks in person.  |
| P60 | secret key on floppy meet to exchange  |
| P56 | Trading public keys on floppy disks would be appropriate for this level of security.   |
| P55 | We could trade disks with our public keys on them. From what I understand, that is the most secure way.  |
| P51 | first, get the lawyer's public key by meeting in person. Once I get his public key, I encrypt it. The same for my lawyer. I and my lawyer must use our secret key to decrypt it. To verify that the message has not been changed, we use our secret key to digitally sign it, and to verify it was send for that person we use the other people (in this case, my lawyer's public key) to verify the signature |
| P52 | I will meet my lawyer in person to exchange the public keys. I will see him in person.   |
| P46 | I will meet him personally and exchange the keys   |

### Basic Lime metaphors

|     |   |
|-----|---|
| P67 | Exchange on floppy in person.   |
| P54 | I believe meeting in person, with your public keys on disk would be appropriate.  |
| P68 | Since the situation deals with one's lawyer the level in which one knows each other is much higher than with some other person like one's ISP. One could send information and ask specific questions in order to know if the person that is sending you their public key information is that person (in this case your lawyer). |
| P43 | This would be worth an in-person meeting to exchange keys.  |
| P44 | Hopefully, we could exchange disks up front, otherwise, my lawyer sending me his key via e-mail would be sufficient. I could then envelope lock him a key of mine and begin messaging securely.   |
| P50 | First, make sure you can trust the other person. If you can, then you generate keys and exchange public + secret key info via disk transferral.   |
| P59 | I need his public key for the envelope lock, as well as my private key for content lock. He would need his private key to open envelope locked message + my public key to check content lock.   |
| P57 | Here again, I might have to fly into his city, so we can make the exchange of public keys in private.   |

P63 Just hand in my lawyer the public key

### Extended Lime metaphors

P45 Using generate-keys function, make a pair of keys. Meet in person with the lawyer to hand over the public key.

P47 1. generate keys. 2. meet, exchange media w/public keys. 3. use keys to verify.

P48 - make a key pair (both lawyer and myself) - each personalize secret key with password - trade public keys

P49 Our public keys would need to be swapped on floppy disks and then those keys would be used to lock and open the signature locks.

P53 yes. meet in person + trade disks.

P58 he would need my public key. I could email it, post online, or give it to him via disk.

P64 we would definitely want to exchange keys on floppies in person because of the sensitive nature of the pending case but because the circumstances of the case are known by both parties it may be possible to exchange public keys via email since key details could be mention to prove the emailer's identity.

P65 Even if I already had a set of keys, I would create a new set, so that only my lawyer has my public key. I would meet with my lawyer and give him a floppy disk with a copy of my public key on it.

P62 a secret key so only lawyer can open

### (16) Are there any comments you would like to make?

#### PGP metaphors

P66 I'm not sure when to use the pass phrase or if I need to once it's established and do the secret keys match

P61 <blank>

P60 I think it would be easier to meet face to face or take the chance on using regular email. Going through all the security channels seems quite cumbersome.

P56 <blank>

P55 <blank>

P51 <blank>

P52 The lawyer should be trustworthy.

P46 I am not sure if the software is that secure, so that I can trust it for such email. I am mainly worried about my public key. Although I think knowing my public key alone will not help anyone.

#### Basic Lime metaphors

P67 This sort of security use would be valuable to me. I have clients who are afraid to communicate by email or cell phone for security reasons.

P54 <blank>

|     |   |
|-----|---|
| P68 | <blank>   |
| P43 | <blank>   |
| P44 | Nah. I have said a lot. Would be interesting to see the software introduced. See how popular it is, that sort of thing. |
| P50 | There should be ways to create secret keys for different groups or b-boards w/o giving out one's main secret key.       |
| P59 | <blank>   |
| P57 | <blank>   |
| P63 | <blank>   |

### Extended Lime metaphors

|     |   |
|-----|---|
| P45 | <blank>   |
| P47 | <blank>   |
| P48 | I just realized I answered the previous questions inaccurately - didn't read the questions carefully. |
| P49 | <blank>   |
| P53 | <blank>   |
| P58 | No.   |
| P64 | <blank>   |
| P65 | <blank>   |
| P62 | Not sure about key but envelope lock would be good in this situation                                  |

### Question 5

You have started a small company, with about 30 employees, which is busy developing a new product line, the details of which must be kept secret from the public and from your competitors. Your employees all need to communicate regularly with each other by email to keep each other up to date on the product strategy and progress. You are hiring additional people at the rate of one or two per week, and the new people need to be integrated into the email communications as quickly as possible.

(17) Would you, in real life, think it was worth putting in some extra time to make these messages secure, rather than simply relying on regular email? If yes, how much extra time (in seconds, minutes, hours, or days) would you think it was worth?

### PGP metaphors

|     |  |
|-----|--|
| P66 | 60 seconds   |
| P61 | <u>yes</u> 10 minutes each new employee                  |
| P60 | 2-3 hours in training. 10-15 in sending message.         |
| P56 | Yes. 1 min.  |
| P55 | I would put in a few days trying to secure our messages. |
| P51 | Yes, hourse  |

P52 15-30 minutes per person.

P46 Yes, a month or so.

### Basic Lime metaphors

P67 Same as prior

P54 Yes, I would spend a couple days to ensure new employees were set up properly and old employees were updated with all the new employees information.

P68 Yes, in this case I would suggest putting in about 14 days of extra time.

P43 Yes, 10 minutes

P44 Still think its important, but 5-10 minutes tops on the time. They gotta get working to produce our product line before someone else does.

P50 YEARS

P59 Yes, I would put days worth of work into security.

P57 yes, possibly a few days

P63 Yes, in 1 minute

### Extended Lime metaphors

P45 Yes. 5 min.

P47 Yes. I would spend an extra 10 min/day.

P48 5 hrs

P49 Yes, no significant amount, maybe 15 seconds a message.

P53 yes, 1 hr.

P58 Yes. I'd say about 10 minutes.

P64 yes - hours or possibly even days

P65 Yes. I would spend a few hours securing the messages.

P62 Yes, a few days

(18) If you answered "yes" to question 17, then can you tell, from the software description you were given, which keys and which functions you and your employees would each need to use? If yes, please list them.

### PGP metaphors

P66 Use receipts public key, read secret key (access to pass phrase). Use your secret to sign (using pass phrase to access), verify identity? with public key

P61 They + I would need to exchange public keys all around. We would encrypt messages, possibly dig. signing. if someone left the company.

P60 public keys for the group

P56 You: encrypt, employee's public key, digital sig. Employees: decrypt, secret key, verify-sig.

P55 My employees would need to use the generate-keys function and they would

|     |   |
|-----|---|
|     | have to know everyone else's public keys.   |
| P51 | public and secret keys, encrypt and decrypt function, digital and verify signature  |
| P52 | I need: (1) My employees public key to encrypt my mails to him (2) my secret key to sign it. My employee needs: (1) my public code to verify the signature (2) A secret code to read my messages. |
| P46 | same as before  |

### Basic Lime metaphors

|     |  |
|-----|--|
| P67 | Same as prior  |
| P54 | The senders of the e-mail would put the receivers public key on there so only a specified person can open it with their private key.   |
| P68 | Again, both types of locks (envelope and content) and use both keys secret and public.   |
| P43 | Content locking - you use your secret key when sending - others verify using your public key   |
| P44 | We would all need our own secret keys and the public keys to everyone in the place. Message lock and content lock would both be employed to assure some outsider doesn't send forgeries. |
| P50 | Public key for all employees. Secret key for anyone faithful longer than 6 months. Content locks on info from me to employees. Envelope locks on sensitive info.                         |
| P59 | All messages would have to be envelope + content locked. We would need to share public keys + have private keys to ourselves.  |
| P57 | public/private, content, envelope  |
| P63 | envelope lock and secret key   |

### Extended Lime metaphors

|     |   |
|-----|---|
| P45 | Envelope lock, content lock & certification lock with public key.   |
| P47 | Each person would need a unique key pair. Mail messages would have content locks and certification locks.   |
| P48 | all employees key pair: secret & public key. could use certification, envelope, content & signature lock  |
| P49 | I would use content locks w/public keys. Forgery of keys wouldn't be a problem since all of the keys would be use internally within the company.  |
| P53 | yes. me: employee's public key - env lock. employees: secret key to open env. lock  |
| P58 | - I would need everyone's public key to envelope lock the message. - everyone would need their secret key to read the message. - everyone else would need everyone's public keys so they can send mail to each other.               |
| P64 | yes - everyone needs their own set of keys as well as everyone else's public keys. all emails would need to be content + envelope locked - public key of the sender used to verify no tampering + secret key of sender used to open |
| P65 | Definitely use an envelope lock. Also, a content lock and a signature lock. Each employee would need his own set of keys. Everyone's public keys would be traded with all.  |
| P62 | Envelope lock so only people in company can open. Public key.   |

(19) If you answered “yes” to question 17, then can you tell, from the software description you were given, what steps you and your employees would each need to take to get those keys at an appropriate level of security? If yes, please list them.

### PGP metaphors

|     |   |
|-----|---|
| P66 | no  |
| P61 | They would be given disks at work in person. If someone left, pass phrases would be changed.  |
| P60 | secret keys, public keys  |
| P56 | Trading keys via floppy disk would be appropriate. the disk could contain others in the company and their public keys could be considered as secure as the person you've traded with.   |
| P55 | I guess you would need to make sure everyone knew everyone elses public keys. Since my old employees trust me + know my public key, I could email them the new employee's public key as I got it. I'd get it from the employee on a disk. Then I could email the new employee all the other employees' public keys. The new employee would have to trust me.  |
| P51 | Meet with each person to receive their public key, and each person receive my public key. to protect an information, I use the person (or people) public key to encrypt it. To read an inf send to me I use my secret key. to make sure that my message has not been changed or forged, I use my secret key to digitally sign it. to verify a message, that has been sent to me, to verify the identity I use his public key to verify-signature on it. |
| P52 | I will exchange public keys in person and hold the employee responsible for any leak of information due to his negligence with secret codes.  |
| P46 | Whoever is recruited to the company will be given the public key, which will be changed every month or two and will be told to them in a joint meeting.   |

### Basic Lime metaphors

|     |  |
|-----|--|
| P67 | Same as prior  |
| P54 | I believe the most effective way would be for the employees to give their private keys to myself (the Boss) and I would be the secure middle man to make sure the appropriate people had received the proper keys.   |
| P68 | Since the situation is of great importance to the company and being a small company (one headquarters in which all employees are in one building, one location) the idea of physically handing them a disk with the necessary information would apply in this situation. |
| P43 | The best way is to provide new employees with your public key. Then you as the boss can send an email (content locked) that contains others public keys.   |
| P44 | I'd say a Master Disk with all keys could be issued upon employment. The person responsible for the network could message lock new employee keys to the existing workers.  |
| P50 | 1) Generate keys 2) Give out keys to employees on a disk.  |
| P59 | Public keys could be distributed on the company website. Since individuals   |

|     |  |
|-----|--|
|     | are the only ones using private keys, they need to be careful to protect them.                 |
| P57 | I may put the public key on an intranet, where only employees have access to. (i.e. a webpage) |
| P63 | create a pass phrase   |

### Extended Line metaphors

|     |   |
|-----|---|
| P45 | Not exactly.  |
| P47 | 1. Generate key pairs. 2. Exchange (via media) the public keys w/the new person. 3. Use this person's other keys for people.  |
| P48 | make key pairs. have one person personally obtain/exchange public keys. distribute certification lock to all employees.   |
| P49 | Each employee is assigned a key and whenever e-mailing uses content locks. Secret keys would not be practical as that would require every employee to know all other employees.   |
| P53 | yes, trade in person @ work   |
| P58 | - email, online, disk.  |
| P64 | keys would need to be traded in person or by a designated key trader who certifies + passes on employees public keys to new employees   |
| P65 | Each of us would have to own own set of keys. The public keys need to be traded. Since we all work together and see each other every day, we should copy the public keys to a floppy disk and trade each disk with everyone else. When someone new joins, they create their keys, pass their floppy to everyone, and borrow everyone's. |
| P62 | Public keys would need to be sent out to allow all the workers to access files.   |

(20) Are there any comments you would like to make?

### PGP metaphors

|     |   |
|-----|---|
| P66 | what seems to work well with one party involved seems cumbersome with more than one and less secure.  |
| P61 | Would changing pass phrases + digitally signing emails be security enough if an ex employee had our public keys?  |
| P60 | <blank>   |
| P56 | <blank>   |
| P55 | Is it possible to set up a bboard with one set of keys? Like, everyone would know the bboard public key. You would still need everyone's public keys (who's posting to it) so you could verify their signature. |
| P51 | <blank>   |
| P52 | How about this. I am the boss and I automatically change the secret key of my employees often each message they send to me or I send to them, they will be told use a particular secret key each time.          |
| P46 | <blank>   |



### Basic Lime metaphors

|     |  |
|-----|--|
| P67 | <blank>  |
| P54 | The security would probably only be as good as the least trustworthy employee.   |
| P68 | <blank>  |
| P43 | <blank>  |
| P44 | This seems kinda shaky. What if someone quit and took the master disk or key list with them. Competitors then could tap into e-mail and use the keys to read it. |
| P50 | Nope   |
| P59 | <blank>  |
| P57 | <blank>  |
| P63 | <blank>  |

### Extended Lime metaphors

|     |  |
|-----|--|
| P45 | <blank>  |
| P47 | <blank>  |
| P48 | oops. did I have signature lock on the lawyer question. because I should have included it. |
| P49 | <blank>  |
| P53 | <blank>  |
| P58 | No.  |
| P64 | are there any restrictions to envelope locking an email for numerous individuals?          |
| P65 | <blank>  |
| P62 | Also good for this situation.  |

### Question 6

You are arranging to have some work done on a house that you own, but you have to be out of town at the time when the contractor is ready to submit a signed bid to you. You don't have a fax readily available, so you want the contractor to submit the bid to you in an email message, but you also need the message to be evidence that the contractor legally signed the bid.

(21) Would you, in real life, think it was worth putting in some extra time to make these messages secure, rather than simply relying on regular email? If yes, how much extra time (in seconds, minutes, hours, or days) would you think it was worth?

### PGP metaphors

|     |                          |
|-----|--------------------------|
| P66 | no, I would not use this |
| P61 | yes 20 minutes           |
| P60 | yes 10-15 minutes        |

|     |  |
|-----|--|
| P56 | Yes. 1 hour.   |
| P55 | In real life, I would do this in person, but if I had to, I'd spend a day on making it secure. |
| P51 | Yes, minutes   |
| P52 | An hour!   |
| P46 | Yes, a day or two  |

### Basic Lime metaphors

|     |   |
|-----|---|
| P67 | 30 minutes to learn, 30 seconds to use.                               |
| P54 | Yes, an hour or so.   |
| P68 | Yes, I would put about 7 to 14 days of extra time in this situation.  |
| P43 | Yes, 10 minutes   |
| P44 | Yes definitely worth the effort. Back to the 20-30 minute time frame. |
| P50 | Couple hours  |
| P59 | Yes. An hour.   |
| P57 | yes, maybe a few minutes  |
| P63 | yes, in 30 seconds  |

### Extended Lime metaphors

|     |   |
|-----|---|
| P45 | Yes. 30 min.  |
| P47 | yes. 30 minutes or so.  |
| P48 | 40 min  |
| P49 | Yes. Around two minutes to assure the contractual bid is legal.                                   |
| P53 | yes 5 min   |
| P58 | I might spend a minute or 2 on this one. I probably would just have someone else pick up the bid. |
| P64 | yes - 10 minutes  |
| P65 | I would put in some extra time, but only up to an hour or two.                                    |
| P62 | Yes a few hours.  |

(22) If you answered “yes” to question 21, then can you tell, from the software description you were given, which keys and which functions you and your contractor would each need to use? If yes, please list them.

### PGP metaphors

|     |  |
|-----|--|
| P66 | <blank>  |
| P61 | I would give my public key to a notary public and have them sign with the contractor and send it digitally signed by the notary. |
| P60 | public key to contractor, verify his signature   |
| P56 | You: secret key, decrypt, verify-sig. Contractor: my public key, encrypt, digital sig.   |

|     |  |
|-----|--|
| P55 | If all I want is to verify that my contractor signed the bid, I would just need to get from him his public key. I could use that to verify-signature.  |
| P51 | encrypt and decrypt function, verify and digital signature function, and public and secret keys  |
| P52 | I need (1) contractor's public key to verify that the message is real. (2) my secret key to read the contractor's encrypted message. My contractor would use (1) my public key to encrypt the message (2) a secret key to sign it. |
| P46 | same as before   |

### Basic Lime metaphors

|     |   |
|-----|---|
| P67 | You would need to be absolutely sure that the user was the contractor if you have prior agreement that use of the system constituted a binding agreement, equivalent to a signature on a printed document. If so, public keys should be exchanged in advance, with signature agreement. Use content lock to protect message and signify content in same way as signature. |
| P54 | The contractor would use his private key and I would use his public key.  |
| P68 | I would provide the mail message with both locks, envelope and content plus use secret and public key.  |
| P43 | The contractor should content lock the message with his secret key and also envelope lock it with your public key.  |
| P44 | We would need our own secret keys and I would need his public key. I don't think its necessary to envelope lock this, just to content lock it. That's why I only need his key.  |
| P50 | Public key, content lock  |
| P59 | Only a content lock b/c you can't have message or signature forged. No envelope lock needed b/c eavesdropping isn't a concern.  |
| P57 | public/private, content/envelope. content lock will be very important.  |
| P63 | an envelope lock & public key   |

### Extended Lime metaphors

|     |  |
|-----|--|
| P45 | Content lock & signature lock with public key & secret key.  |
| P47 | 1. a unique public/private key pair. 2. a content lock (to prevent later tampering), a signature lock.   |
| P48 | envelope lock, content, signature lock. use key pair: public & secret.   |
| P49 | A signature lock would insure authenticity, and a public key would provide enough security.  |
| P53 | contractor: secret key - signature lock. me: contractor's public key -> check signature lock.  |
| P58 | - The contractor would need to put a signature lock on the mail. - Assuming that signature lock falls under content lock, they would need their secret key. - I would need their public key to read the message.   |
| P64 | yes - both parties need their own set of keys + each other's public keys. actually the contractor would only need to give out their public key since the email doesn't need to be envelope locked. the email from the contractor would need to be only signature locked to protect the contents + to be a legal signature. |
| P65 | I would definitely use a content lock. Maybe an envelope lock. I could make a set of keys and give my contractor my public key.  |

P62 Signature lock to prove who it is from, with a secret key b/c you and contractor are only 2 people to open.

(23) If you answered "yes" to question 21, then can you tell, from the software description you were given, what steps you and your contractor would each need to take to get those keys at an appropriate level of security? If yes, please list them.

### PGP metaphors

P66 <blank>

P61 I would take the disk with my public key to the notary in person.

P60 public key, secret key, trade key

P56 You would need to trade keys on floppy disk to achieve appropriate level of security.

P55 I assume that I have seen him in person before, so we could exchange disks with our public keys. If I only want to verify that he sent the email, I would just need his public key - he wouldn't need mine.

P51 I send an email with my public key, as same as the contractor. To protect the message, I would use the contractor's public key to encrypt it. To read his message, I would use my private key to decrypt. To protect the message for being forged, each would use our secret key to digitally sign it. To verify that the message has been send for that person I would use his public key to verify-signature.

P52 Will meet in person and get a legal document signed from him that he uses specific secret and public keys and he is responsible for the message sent using them!!

P46 I will talk to him on phone and can exchange public keys.

### Basic Lime metaphors

P67 Same as prior cases where disks were exchanged. Downloading would be unsatisfactory.

P54 Have the contractor hand you a copy of his public key the first time he comes to your home.

P68 Since one is out of town the idea of physically handing a disk of information is out of the question. The solution for this particular situation would be to consider the possibility of asking question about the contractor that only the contractor would know. One must develop a close relationship prior to the business transaction.

P43 This would best be done with an in-person meeting to exchange public keys.

P44 Assuming I didn't get it from him earlier, I could make sure his e-mail address is the same as before, or ask him specific questions to verify it came from him and have him e-mail it to me.

P50 Generate keys. Exchange public keys via telephone or however. Use content lock to make sure no one tampers with contractor's email to you.

P59 He would use his secret key for a content lock. He would need to send his public key to me so that I could check the content lock.

P57 I would arrange a meeting ahead of time, (when we discuss what needs to be done), and get his public key then.

P63 | just give the contractor the public key

### Extended Lime metaphors

P45 | Not really.

P47 | 1. generate keys. 2. get his public key from a website.

P48 | each have key pair. meet & exchange public key.

P49 | The contractor would need to signature lock the email and then send it. I would access it using the public key.

P53 | yes, trade in person

P58 | - email, online, floppy. he would have his secret lock.

P64 | the contractor should put his public key on his web site for easy access.

P65 | I would make the set of keys, copy the public key to a floppy disk, and give him the disk before I left town.

P62 | Need to send a certification lock with an envelope locked message, so with the secret key only the you and the contractor can open.

(24) Are there any comments you would like to make?

### PGP metaphors

P66 | <blank>

P61 | Are we allowed to add extra people?

P60 | <blank>

P56 | <blank>

P55 | If I traded my public key w/contractors or online businesses when shopping, I bet they would start selling them to solicitors like they do addresses, phone #'s + email addresses.

P51 | <blank>

P52 | ~~Look tricky!~~ The

P46 | I am not sure about how much imp. a signed bid is?

### Basic Lime metaphors

P67 | <blank>

P54 | <blank>

P68 | <blank>

P43 | <blank>

P44 | e-mailing public keys unsealed seems unsafe. Couldn't someone intercept one and masquerade as you? well, they could send private mail as you I guess

P50 | Think about ability to create group public keys w/secret keys to certain sections.

P59 | <blank>

P57 | <blank>

P63 <blank>

### Extended Lime metaphors

P45 <blank>

P47 <blank>

P48 <blank>

P49 <blank>

P53 <blank>

P58 No.

P64 <blank>

P65 <blank>

P62 A little confusing on exact method for this one, but idea sounds interesting.

### Debriefing questionnaire

1) Do you think the description provided the right amount of information about how the security worked? (circle one)

- (a) Not nearly enough information
- (b) Almost enough information
- (c) Just the right amount of information
- (d) Slightly too much information
- (e) Way too much information

### PGP metaphors

P66 b

P61 b

P60 e

P56 c

P55 b

P51 c

P52 c

P46 c

### Basic Lime metaphors

P67 b

P54 b

P68 b

P43 c

P44 c

|     |   |
|-----|---|
| P50 | b |
| P59 | d |
| P57 | b |
| P63 | c |

### Extended Lime metaphors

|     |   |
|-----|---|
| P45 | a |
| P47 | c |
| P48 | c |
| P49 | c |
| P53 | c |
| P58 | c |
| P64 | b |
| P65 | c |
| P62 | b |

## 2) What was the most confusing part of the description?

### PGP metaphors

|     |   |
|-----|---|
| P66 | Trading public keys and access to these keys and its changeable public secret   |
| P61 | assuming a layperson would understand the exchange of intangible objects like "a public key". Is that a password or an id or something? |
| P60 | the trading of the keys became confusing to me and having to keep adding on to get more security  |
| P56 | Trading public keys to get basic security.  |
| P55 | The description wasn't confusing, but it needed more information.   |
| P51 | the secret and public key related of how you can trade them   |
| P52 | Trading to get the keys for stronger security.  |
| P46 | Trading public keys   |

### Basic Lime metaphors

|     |   |
|-----|---|
| P67 | Description was clear enough, but could profit from more graphics and examples.   |
| P54 | Understanding which key to put on a message isn't confusing, but it just takes a few glances before you can remember which key performs which lock. |
| P68 | Trading public keys to get basic security & keys  |
| P43 | The description of how others might try to obtain your public key.  |
| P44 | Exactly which keys locked and unlocked which documents. After a time or two, I got the hang of it though.   |
| P50 | The last paragraph when talking about using person A to get person B's info.  |

|     |  |
|-----|--|
| P59 | Trading keys for stronger security.                  |
| P57 | How the attacker set up web pages, or forged emails. |
| P63 | There are too many ways to trade public keys         |

### Extended Lime metaphors

|     |   |
|-----|---|
| P45 | How the keys are obtained & shared  |
| P47 | which keys were necessary for which locks   |
| P48 | ways to get a public key  |
| P49 | The description of keys and how they are used.  |
| P53 | trading keys to get basic/stronger security. certification lock.                      |
| P58 | the types of keys needed to open the specific locks was hard to keep straight.        |
| P64 | what to do with the backup key pair   |
| P65 | Discussing what each key should be used for. Trading public keys.                     |
| P62 | Too many options a little. Might want to narrow down how many combinations there are. |

### 3) What was the clearest part of the description?

#### PGP metaphors

|     |  |
|-----|--|
| P66 | protecting the secret key  |
| P61 | What you would use the functions for.  |
| P60 | the encrypted message part I could follow that   |
| P56 | encrypt-decrypt, digital-sig - verify sig.   |
| P55 | The first page and beginning of the second - the function descriptions. "Security functions" and "Keys". |
| P51 | the functions  |
| P52 | keys describes on page 2.  |
| P46 | The use of encrypt & decrypt fn  |

#### Basic Lime metaphors

|     |   |
|-----|---|
| P67 | See above.  |
| P54 | Proper ways to trade keys.  |
| P68 | Locks & trading public keys to get stronger security                |
| P43 | The <u>keys</u> section was very clear and easy to reference.       |
| P44 | The description of the locks and what each lock did.                |
| P50 | The part with pictures.   |
| P59 | Keys + locks; i.e. how to read + check locks of email               |
| P57 | How to read a message, or to put a lock on it. (bulleted sentences) |
| P63 | There are two pictures to describe the functions of the locks       |



### Extended Lime metaphors

|     |  |
|-----|--|
| P45 | The function of different types of locks.                  |
| P47 | how the keys are distributed                               |
| P48 | different kinds of locks                                   |
| P49 | The section on locks is very straight forward.             |
| P53 | 1st page: locks  |
| P58 | the descriptions of the locks were clear.                  |
| P64 | the locks description + the outline of how to use the keys |
| P65 | "Protecting your secret key", but it was all pretty clear. |
| P62 | Description of locks, but not keys.                        |

#### 4) What question about the description would you most like answered?

### PGP metaphors

|     |   |
|-----|---|
| P66 | Does the code phrase change or is it a constant. Does your secret key change for different situation, persons |
| P61 | please see #2!  |
| P60 | Is this necessary for the average person or is it more for people doing extremely confidential work           |
| P56 | Don't understand the ways to transfer keys for basic security.  |
| P55 | Can I make more than one set of keys?   |
| P51 | the functions and what are the secret and public keys   |
| P52 | How is it different from the regular concept of a login name (public key) and password (secret key)           |
| P46 | How much difficult is it to forge a digital signature?  |

### Basic Lime metaphors

|     |   |
|-----|---|
| P67 | How do I know that security of messages I have sent has not been compromised?   |
| P54 | Would you be able to put multiple locks on a message if you were sending it to a group of people?   |
| P68 | In general, how can people get into your mail message?  |
| P43 | I think a bit more might needs to be added about why this security is needed. How easy can someone forge a message?                                 |
| P44 | How exactly you can ensure your secret key is safe? Is it kept on disk and never down loaded? Couldn't a hacker access your system and download it? |
| P50 | Will this software be affordable to the average Jane?   |
| P59 | Why do you need stronger security?  |
| P57 | <arrow to #2>   |
| P63 | I will list a simple procedure steps about how to trade the keys.   |

### Extended Lime metaphors

|     |   |
|-----|---|
| P45 | The time you are willing to spend to secure a particular kind of message. |
| P47 | none  |
| P48 | do both parties need this software?                                       |
| P49 | when and how to use what key  |
| P53 | explain certification lock better   |
| P58 | what if a stranger needed to send you email? how could you read it?       |
| P64 | <blank>   |
| P65 | Secure ways to trade your public key without having to use a floppy disk. |
| P62 | Why some many keys? How about one key?                                    |

5) Do you think you would make use of security software like this for your email messages if it was available? (circle one)

- (a) Not at all
- (b) Once in a while
- (c) Frequently
- (d) Almost always
- (e) Always

### PGP metaphors

|     |   |
|-----|---|
| P66 | c |
| P61 | b |
| P60 | b |
| P56 | b |
| P55 | c |
| P51 | b |
| P52 | b |
| P46 | b |

### Basic Lime metaphors

|     |   |
|-----|---|
| P67 | c |
| P54 | b |
| P68 | e |
| P43 | b |
| P44 | c |
| P50 | d |
| P59 | a |
| P57 | d |
| P63 | b |

### Extended Lime metaphors

|     |   |
|-----|---|
| P45 | b |
| P47 | b |
| P48 | a |
| P49 | b |
| P53 | e |
| P58 | a |
| P64 | b |
| P65 | b |
| P62 | b |

### 6) Do you have any other comments you'd like to make?

#### PGP metaphors

|     |   |
|-----|---|
| P66 | I think the use of public is confusing many people would assume it is public and open to all. I think a more useful tool such as private or personal might work better. As it stands the description would not induce me to use software. It would not suggest the security I would need! |
| P61 | Overall the description was very clear + concise. I was confused on the points I brought up on the last question of the 1st set. That is, what exactly is being exchanged and does the message remain securely digitally signed after verification?                                       |
| P60 | A security software that has less steps might be better   |
| P56 | <blank>   |
| P55 | Getting keys and verifying you are getting keys from the right person sounds like a huge hassle.  |
| P51 | <blank>   |
| P52 | For security as suggested, I still have to interact with the person and ask personal questions which I can anyway do on regular email. what's the big advantage here?   |
| P46 | Maybe a few more explanations are needed to trust the software.   |

#### Basic Lime metaphors

|     |  |
|-----|--|
| P67 | "Each person who uses the security software must have" his/her own matched keys, not "their". This error appears several places in the questions, too. |
| P54 | <blank>  |
| P68 | <blank>  |
| P43 | <blank>  |
| P44 | it's available when?   |
| P50 | Use step-by-step pictures.   |
| P59 | It seems sort of a complex system for protecting email privacy. Being careful to protect keys could be complicated sometimes. Key distribution         |

|     |                                   |
|-----|-----------------------------------|
|     | could be somewhat confusing also. |
| P57 | <blank>                           |
| P63 | <blank>                           |

### Extended Lime metaphors

|     |   |
|-----|---|
| P45 | <blank>   |
| P47 | <blank>   |
| P48 | <blank>   |
| P49 | I imagine the software would be easier to understand if I were using it and not just reading about it.  |
| P53 | put the keys + locks into a chart that might make it easier. examples are good.   |
| P58 | this system seems impractical   |
| P64 | <blank>   |
| P65 | I generally don't care if others read my email because it never says anything important. But, if I used it to contact a lawyer, I would definitely want it to be more secure. |
| P62 | Interesting if we are dealing with very confidential files for example government but not for ordinary emails.  |

## **APPENDIX D Lime user test materials and data**

This appendix includes the following test materials and data from the Lime software user test:

- D.1 Participant intake questionnaire
- D.2 Participant initial briefing materials
- D.3 Scenario descriptions given to participants
- D.4 Post-test questionnaire
- D.5 Collated results for true/false questions

I have not included the transcripts of the test sessions here because they would increase the size of this document by several hundred pages; instead, I will make them available separately upon request.

## D.1 Participant intake questionnaire

Thank you for your interest in participating in the testing! Here is the intake questionnaire. The answers will be used to select a set of test participants that has the particular demographic characteristics needed for this research study. All information you give will be kept private, and will only be included in research results in anonymized form.

1. Are you a U.S. citizen? If not, do you have an employment authorization card? (If the answer to both those questions is no, then unfortunately I am prohibited from using you for the testing because of the payment.) If you are not a U.S. citizen, but you do have a green card, then I will need to get a photocopy of your green card at the time of the testing, due to university regulations.
2. How old are you?
3. What is your highest education level (high school, some college, undergrad degree, some grad school, grad degree)?
4. What is your profession or main area of expertise (for example: arts, science, medicine, business, engineering, computers, administration...)?
5. For how long have you been using electronic mail?
6. Have you ever studied number theory or cryptography?
7. Have you ever used security software, such as secure email in Netscape or Microsoft Outlook, or PGP, or any other software that involved data encryption? If yes, what was the name of the software?
8. Do you know the difference between public (asymmetric) key cryptography and private (symmetric) key cryptography? If yes, please explain briefly.

Thanks again, and I look forward to hearing from you.

## D.2 Participant initial briefing materials

### Lime Secure Electronic Mail Test: Set-up info

Here is the information you should enter when setting up Lime:

Your full name: enter anything you like here

Your email address: tester@lime.org

Your mail server: lime.org

## D.3 Scenario descriptions given to participant

### Lime Secure Electronic Mail Test: Scenario #1

For the first part of this test, please imagine that you have been seeing articles in the news about how insecure email is, and that you have become curious about software products that offer to protect your privacy on-line. You have acquired a copy of Lime, which is a free software program that is supposed to protect your email, and you want to try it out.

You decide to try sending secure email to your friend Steve. You and Steve have been friends ever since you were kids, and you have fond memories of assembling giant rock collections together when you were ten. You get along really well with his wife Laura, too, although there was a tense moment when you broke one of her favorite wine glasses. Steve works for an advertising agency these days, and you usually use his address there when you email him: [steve@highconcept.com](mailto:steve@highconcept.com).

You have a copy of Lime on your computer. Please use it to send a private, unforgeable email message to Steve. You will need to do some set-up, and you may also receive email that you need to respond to. The test monitor will let you know when the scenario ends.



## Lime Secure Electronic Mail Test: Scenario #2

For the second part of this test, please imagine that you have decided to do volunteer work for a political campaign. The campaign manager, Maria Simmons, has given you the job of campaign coordinator. It is your responsibility to keep the campaign team members up to date on all aspects of the campaign plan.

You will use Lime to communicate with the campaign team members by email. It is very important that no information about the campaign plan gets leaked to the media or to the opposing campaigns. You will therefore need to be very careful to make sure that all your email messages are as private and unforgeable as you can make them.

You have a floppy disk that Maria gave you with her public key on it, and you gave Maria your public key on a floppy disk at the same time. Maria also gave you a printed memo that contains the first campaign plan update.

The campaign team members are:

Ben Dawson (daws@camp2002.org)  
Judy Rivera (judy@camp2002.org)  
Sam Tyler (samt@camp2002.org)  
Maria Simmons (manager@camp2002.org)

Please send the update information to all of the campaign team members in a private, unforgeable email. When you have done that, follow the directions in any email you receive from a campaign team member. The test monitor will let you know when the scenario ends.

## MEMO

TO: Campaign coordinator  
FROM: Maria Simmons

Please send this update information out to the members of the campaign team. Remember to use secure, unforgeable email. Thanks!

Update information as follows:

New dates and times for speeches:

March 7, San Jose  
March 9, Palo Alto  
March 14, Sacramento

## Lime Secure Electronic Mail Test: Scenario #3

For the third part of the test, please imagine that you are still volunteering for the political campaign from Scenario #2, but that the campaign manager has decided to give you some additional responsibility. Your job now includes approving requisitions that are emailed to you by members of the campaign team, as long as they do not exceed \$100 for any one campaign team member.

Here are your detailed instructions for what to do when you receive a requisition by email:

1. Is the requisition unforgeably signed by the campaign team member who is submitting it? If no, send a reply rejecting the requisition for that reason. If yes, go to 2.
2. Does the amount of this requisition, plus the amounts of any previous requisitions you have approved for this campaign team member, exceed \$100? If yes, send a reply rejecting the requisition for that reason. If no, go to step 3.
3. Approve the requisition by putting your own unforgeable signature on it, and send the approved copy to the campaign team member who submitted it.

Please wait for email messages from the campaign team members and respond to them according to the above directions. The test monitor will let you know when the scenario ends.

## D.4 Post-test questionnaire

### **Lime User Test: Post Test Questionnaire**

Thank you for participating in the user test of Lime! Please use your experience from the test to label the following statements as either true or false:

1. You cannot exchange secure email messages with someone unless each of you has a key pair.
2. You should give your secret key to the people you want to exchange secure email with.
3. You should give your public key only to people you know you can trust.
4. If a message has an envelope lock on it, no-one can read it unless they have the right key.
5. Envelope locks don't tell you anything about who applied the lock.
6. Content locks don't tell you anything about who applied the lock.
7. Content locks don't tell you anything about whether a message has been modified since the lock was applied.
8. If you use my public key to put an envelope lock on a message, then no-one can read that message unless they have my matching secret key.
9. If I want to put an envelope lock on a message so that only you can read it, I should use my secret key.
10. Content locks that you make are only useful to people who have your public key.
11. If you want to put your legal signature on a secure message, you need to use both a content lock and a signature lock.
12. You can tell whether you have the right public key for someone by checking the name and email address attached to it.
13. You can trade public keys by email and be fairly sure that you are getting the right public key, as long as both people include personal messages that talk about things a stranger wouldn't know.
14. Putting your certification lock on a public key means that you are the owner of that public key.

### **Additional questions**

1) You are emailing your doctor to tell her or him some information about your medical history. Which of these most accurately describes the security you would want for your email message?

No need for security

Would want protection against random eavesdroppers/forgers, but not worried about a targeted attack.

Would want the strongest possible protection.

Do you think you would be able to use Lime to get the security you would want, or do you think it would be too difficult, or too much work?

2) You are emailing the IRS to tell them some of your tax information. Which of these most accurately describes the security you would want for your email message?

No need for security

Would want protection against random eavesdroppers/forgers, but not worried about a targeted attack.

Would want the strongest possible protection.

Do you think you would be able to use Lime to get the security you would want, or do you think it would be too difficult, or too much work?

3) You are emailing a friend to suggest movie showtimes for a weekend outing. Which of these most accurately describes the security you would want for your email message?

No need for security

Would want protection against random eavesdroppers/forgers, but not worried about a targeted attack.

Would want the strongest possible protection.

Do you think you would be able to use Lime to get the security you would want, or do you think it would be too difficult, or too much work?

4) You are emailing a merchant to give them your credit card information. Which of these most accurately describes the security you would want for your email message?

No need for security

Would want protection against random eavesdroppers/forgers, but not worried about a targeted attack.

Would want the strongest possible protection.

Do you think you would be able to use Lime to get the security you would want, or do you think it would be too difficult, or too much work?

5) You are emailing a family member to remind them to buy bread. Which of these most accurately describes the security you would want for your email message?

No need for security

Would want protection against random eavesdroppers/forgers, but not worried about a targeted attack.

Would want the strongest possible protection.

Do you think you would be able to use Lime to get the security you would want, or do you think it would be too difficult, or too much work?

6) Are there any other comments you would like to make?

## D.5 Collated results for true/false questions

You cannot exchange secure email messages with someone unless each of you has a key pair.

Correct (T): P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14  
Incorrect (F): P1

You should give your secret key to the people you want to exchange secure email with.

Correct (F): P4, P5, P6, P8, P10, P11, P12, P13  
Incorrect (T): P1, P7, P9, P14

You should give your public key only to people you know you can trust.

Correct (F): P5, P7, P9, P10, P12, P13, P14  
Incorrect (T): P1, P4, P6, P8, P11

If a message has an envelope lock on it, no-one can read it unless they have the right key.

Correct (T): P1, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14  
Incorrect (F):

Envelope locks don't tell you anything about who applied the lock.

Correct (T): P5, P6, P7, P8, P9, P10, P12,  
Incorrect (F): P1, P4, P11, P13, P14

Content locks don't tell you anything about who applied the lock.

Correct (F): P1, P4, P5, P6, P7, P9, P10, P12, P13, P14  
Incorrect (T): P8, P11

Content locks don't tell you anything about whether a message has been modified since the lock was applied.

Correct (F): P1, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14  
Incorrect (T):

If you use my public key to put an envelope lock on a message, then no-one can read that message unless they have my matching secret key.

Correct (T): P1, P5, P6, P7, P10, P12, P14  
Incorrect (F): P8, P9, P11, P13  
\*\*\*Other: P4 answered as "false, as other approved keys can also read"

If I want to put an envelope lock on a message so that only you can read it, I should use my secret key.

Correct (F): P4, P5, P6, P8, P12, P13,  
Incorrect (T): P1, P7, P9, P10, P11, P14

Content locks that you make are only useful to people who have your public key.

Correct (T): P1, P5, P8, P9, P13  
Incorrect (F): P4, P6, P7, P10, P11, P12, P14

If you want to put your legal signature on a secure message, you need to use both a content lock and a signature lock.

Correct (F): P6, P10, P11,  
Incorrect (T): P1, P4, P5, P7, P8, P9, P12, P13, P14

You can tell whether you have the right public key for someone by checking the name and email address attached to it.

Correct (F): P1, P4, P5, P6, P7, P8, P9, P12, P13, P14  
Incorrect (T): P10, P11

You can trade public keys by email and be fairly sure that you are getting the right public key, as long as both people include personal messages that talk about things a stranger wouldn't know.

Correct (T): P1, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14  
Incorrect (F):

Putting your certification lock on a public key means that you are the owner of that public key.

Correct (F): P1, P4, P5, P6, P7, P9, P10, P11, P12, P13,  
Incorrect (T): P8, P14



## References

- [Adams00] Anne Adams. Multimedia information changes the whole privacy ballgame. In *Proceedings of the Conference on Computers, Freedom and Privacy 2000*, ACM Press.
- [Adams99] Anne Adams. The implications of users' privacy perception on communication and information privacy policies. In *Proceedings of Telecommunications Policy Research Conference*, Washington, DC, 1999.
- [Adams99-2] Anne Adams and Martina Angela Sasse. Privacy issues in ubiquitous multimedia environments: Wake sleeping dogs, or let them lie? In *Proceedings of INTERACT '99*, Edinburgh, pp. 214—221.
- [Adams99-3] Anne Adams and Martina Angela Sasse. Users are not the enemy: Why users compromise security mechanisms and how to take remedial measures. *Communications of the ACM*, 42 (12), pp. 40—46, December 1999.
- [Ammenwerth99] Elske Ammenwerth, Anke Buchauer, Hans-Bernd Bludau, Alexander Roßnagel. Simulation studies for the evaluation of security technology. *Multilateral Security in Communications, Vol. 3 – Technology, Infrastructure, Economy*. Guenter Mueller and Kai Rannenber (Eds.), Addison-Wesley, 1999.
- [Anderson94] Ross Anderson. Why Cryptosystems Fail. *Communications of the ACM*, 37(11), 1994.
- [ANSI98] American National Standards Institute. *ANSI Z535.4 Product Safety Signs and Labels*, 1998.
- [Apple96] Apple Computer. *Macintosh Human Interface Guidelines*. 1996.
- [Balfanz00] Dirk Balfanz and Dan Simon, WindowBox: A Simple Security Model for the Connected Desktop. In *Proceedings of the 4<sup>th</sup> USENIX Windows Systems Symposium*, August 2000.
- [Brostoff00] Sacha Brostoff and Martina Angela Sasse. Are passfaces more usable than passwords? A field trial. In S. McDonald, Y. Waern & G. Cockton [Eds.]: *People and Computers XIV - Usability or Else!* Proceedings of HCI 2000 (September 5th - 8th, Sunderland, UK), pp. 405-424. Springer.
- [Brown99] Ian Brown and Richard Snow. A proxy approach to e-mail security. *Software - Practice and Experience*, 29(12) 1049-1060, October 1999.

- [Carroll84] J.M. Carroll and C. Carrithers. Training Wheels in a User Interface. *Communications of the ACM*, Vol. 27(8):pages 800--806, August 1984.
- [Carroll85] Carroll, J.M. and Mack, R.M. Metaphor, computing systems, and active learning. *International Journal of Man-Machine Studies*, 22, 39-58, 1985.
- [Collins97] Collins, J.A., Greer, J.E., Kumar, V.S., McCalla, G.I., Meagher, P., and Tkatch, R. Inspectable User Models for Just-In-Time Workplace Training. In *User Modeling: Proceedings of the Sixth International Conference*. Springer Wien. New York. 1997.
- [Coventry02] Lynne Coventry, Antonella De Angeli and Graham Johnson, *Honest it's me! Self service verification*. Presented at the CHI 2003 Workshop on Human-Computer Interaction and Security Systems, April 2003.
- [Cranor99] Lorrie Cranor and Mark Ackerman. Privacy Critics: UI Components to Safeguard Users' Privacy. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'99)*, short papers (v.2.), p. 258-259.
- [Damker99] Herbert Damker, Ulrich Pordesch and Martin Reichenbach. Personal Reachability and Security Management -- Negotiation of Multilateral Security. *Multilateral Security in Communications, Volume 3 - Technology, Infrastructure, Economy*. Guenter Mueller and Kai Rannenberg (Eds.), Addison Wesley 1999.
- [Davis96] Don Davis. Compliance Defects in Public-Key Cryptography. *Proceedings of the 6th USENIX Security Symposium*, 1996.
- [Dhamija00] Rachna Dhamija. Hash Visualization in User Authentication. In *CHI 2000 Extended Abstracts*, April 2000, The Hague, Netherlands.
- [Dhamija00-2] Rachna Dhamija and Adrian Perrig. Deja Vu: A User Study. Using Images for Authentication. In *Proceedings of the 9th USENIX Security Symposium*, August 2000, Denver, Colorado.
- [Dourish03] Paul Dourish, Jessica Delgado de la Flor, and Melissa Joseph, *Security as a Practical Problem: Some Preliminary Observations of Everyday Mental Models*. Presented at the CHI 2003 Workshop on Human Computer Interaction and Security Systems, April 2003.
- [Dufft99] Cornelius C. Dufft, Juergen Espey, Hartmut Neuf, Georg Rudinger and Kurt Stapf. Usability and Security. *Multilateral Security in Communications, Volume 3 - Technology, Infrastructure, Economy*. Guenter Mueller and Kai Rannenberg (Eds.), Addison Wesley 1999, pp.531-545.
- [Friedman02] Friedman, B., Nissenbaum, H., Hurley, D., Howe, D.C., and Felten, E., Users' conceptions of risks and harms on the Web: A comparative study. *CHI 2002*

*Extended Abstracts of the Conference on Human Factors in Computing Systems* (pp. 614-615).

[Friedman02-2] Friedman, B., Hurley, D., Howe, D.C., Felten, E., and Nissenbaum, H., Users' conceptions of Web security: A comparative study. *CHI 2002 Extended Abstracts of the Conference on Human Factors in Computing Systems* (pp 746-747).

[Friedman02-3] Friedman, B., Howe, D.C., and Felten, E., Informed Consent in the Mozilla Browser: Implementing Value-Sensitive Design. *Proceedings of the Thirty-fifth Annual Hawai'i International Conference on System Sciences*, 2002.

[Fritz98] J. Steven Fritzinger and Marianne Mueller. *Java Security*. Sun Microsystems White Paper, 1998.

[Garfinkel03] Simson Garfinkel, Email-Based Identification and Authentication: An Alternative to PKI? *IEEE Security and Privacy*, November/December 2003.

[Garfinkel03-2] Simson Garfinkel, *Enabling Email Confidentiality through the use of Opportunistic Encryption*. Presented at the 2003 National Conference on Digital Government Research, May 2003.

[Guzdial95] Guzdial, M. Software-realized Scaffolding to Facilitate Programming for Science Learning. *Interactive Learning Environments*, Vol. 4, No. 1, 1995, 1-44.

[Holmström99] Ursula Holmström. User-centered design of security software. *Human Factors in Telecommunications*, May 1999, Copenhagen, Denmark.

[Jendricke00] Uwe Jendricke and Daniela Gerd tom Markotten. Usability meets Security - The Identity-Manager as your Personal Security Assistant for the Internet. In *Proceedings of the 16th Annual Computer Security Applications Conference*, December 2000.

[Jermyn99] Jermyn, I., Mayer, A., Monroe, F., Reiter, M.K., Rubin, A.D. The Design and Analysis of Graphical Passwords. *Proceedings of the 8th USENIX Security Symposium*, August 23-36, 1999.

[Johnson02] Jeff Johnson and Austin Henderson. Conceptual Models: Begin by Designing What to Design. *Interactions*, January & February 2002.

[Karat89] Claire-Marie Karat, Iterative Usability Testing of a Security Application. *Proceedings of the Human Factors Society 33rd Annual Meeting*, 1989.

[Karvonen99] Kristiina Karvonen. Enhancing Trust Online. *Proceedings of PhDIT'99: Ethics in Information Technology Design*. Second International Workshop on Philosophy of Design and Information Technology, 16-17 December, 1999, Saint-Ferréol, Toulouse, France.

- [Kolojejchick97] J. Kolojejchick, S.F. Roth, and P. Lucas, Information Appliances and Tools in Visage. *IEEE Computer Graphics and Applications*, Vol. 17, No. 4, July, 1997, pp. 32-41.
- [Kuhn91] W. Kuhn and A. Frank, A Formalization of Metaphors And Image-Schemas in User Interfaces. In *Cognitive and Linguistic Aspects of Geographic Space* (D.M. Mark & A.U. Frank, eds.) Kluwer Academic Publishers, Netherlands, pp. 419-434.
- [Lau99] Tessa Lau, Oren Etzioni and Daniel S. Weld. Privacy Interfaces for Information Management. *Communications of the ACM*, 42(10), October 1999.
- [Laukka00] Markku Laukka. Criteria for Privacy Supporting System, *Proceedings of the fifth Nordic Workshop on Secure IT systems* (Nordsec 2000), October 12-13, 2000, Reykjavik, Iceland.
- [Leveson95] Nancy G. Leveson. *Safeware: System Safety and Computers*. Addison-Wesley Publishing Company, 1995.
- [Liddle96] David Liddle, Design of the Conceptual Model. Interview by Barry Polley, Andrew Singer, Suzanne Stefanac and Terry Winograd, in *Bringing Design to Software*, Addison-Wesley, 1996.
- [Mandel97] Theo Mandel. *The Elements of User Interface Design*. John Wiley & Sons, February 1997.
- [Nardi93] Nardi, B. and Zamer, C. Beyond models and metaphors: Visual formalisms in user interface design. *Journal of Visual Languages and Computing* 4, 5-33, 1993.
- [Neale97] Dennis C. Neale and John M. Carroll. The Role of Metaphors in User Interface Design. In *Handbook of Human-Computer Interaction, 2<sup>nd</sup> Edition*, M. Helander, T.K. Landauer, P. Prabhu (eds.). North-Holland, 1997.
- [Nielsen94] Jakob Nielsen. Heuristic Evaluation. In *Usability Inspection Methods*, John Wiley & Sons., Inc., 1994.
- [Norman94] Donald A. Norman, *Things that make us smart: Defending human attributes in the age of the machine*. Perseus Publishing, 1994.
- [Shen92] HongHai Shen and Prasun Dewan. Access Control for Collaborative Environments. *Proceedings of CSCW '92*.
- [Sun95] Sun Microsystems. *HotJava Browser: A White Paper*. Sun Microsystems White Paper, 1995.

[Whitten99] Alma Whitten and J.D. Tygar, Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0. *Proceedings of the 9<sup>th</sup> USENIX Security Symposium*, August 1999.

[Whitten98] Alma Whitten and J.D. Tygar, *Usability of Security: A Case Study*. Technical Report CMU-CS-98-155, Carnegie Mellon University School of Computer Science, December 1998.

[Wolf99] Gritta Wolf and Andreas Pfitzmann. Empowering Users to Set Their Protection Goals. *Multilateral Security in Communications, Volume 3 - Technology, Infrastructure, Economy*. Guenter Mueller and Kai Rannenberg (Eds.), Addison Wesley 1999, pp.113-135.

[Ye02] Zishuang Ye and Sean Smith, Trusted Paths for Browsers. *11<sup>th</sup> USENIX Security Symposium*, August 2002.

[Yee03] Ka-Ping Yee, *Secure Interaction Design and the Principle of Least Authority*. Presented at CHI 2003 Workshop on Human-Computer Interaction and Security Systems, April 2003.

[Zurko96] Mary Ellen Zurko and Richard T. Simon. *User-Centered Security*. New Security Paradigms Workshop, 1996.

[Zurko99] Mary Ellen Zurko, Richard T. Simon and Tom Sanfilippo, A User-Centered, Modular Authorization Service Built on an RBAC Foundation. *Proceedings of IEEE Security and Privacy*, 1999.