Moving from the Design of Usable Security Technologies to the Design of Useful Secure Applications

D. K. Smetters
PARC
3333 Coyote Hill Rd.
Palo Alto, CA, USA
smetters@parc.com

R. E. Grinter
PARC
3333 Coyote Hill Rd.
Palo Alto, CA, USA
beki@parc.com

ABSTRACT
Recent results from usability studies of security systems have shown that end-users find them difficult to adopt and use. In this paper we argue that improving the usability of security technology is only one part of the problem, and that what is missed is the need to design usable and useful systems that provide security to end-users in terms of the applications that they use and the tasks they want to achieve. We propose alternate ways of building and integrating security technologies into applications and usability methods for evaluating how successful our prototypes are. We believe that the end results of designing usable and useful (from the end-user perspective) systems will be secure applications which will reflect the needs of users who are increasingly using computers away from the office and in a wider variety of networked configurations.

Categories and Subject Descriptors
D.4.6 [**Operating Systems Software**]: Security and Protection; H.1.2 [**Models and Principles**]: User/Machine Systems—*Human factors*

General Terms
Human Factors, Security

Keywords
Usable security

1. INTRODUCTION
There is a small, but increasingly vocal movement suggesting that usability of security technology may be one of the largest roadblocks standing in the way of increased computer security [29, 1, 26, 27]. Empirical studies have exposed usability problems in password systems [3, 1, 26], access control mechanisms [29, 18, 28], and encryption software [27]. To address these problems, the authors of these studies suggest redesigning applications to be more realistic in what

they expect of users, and education of the users themselves to be more effective participants in system security (*e.g.*, by choosing better passwords). But very little work (*e.g.*, [20, 21, 4]) attempts to design new technologies that solve security problems from a usability perspective.

At the same time, a number of authors have suggested that much "classical" computer security technology is designed for an environment out of step with current computer use and end-user goals [8, 23], and that it may be time for a shift in the design of secure systems [8] in order to provide technology more closely in line with the needs of non-military users. This is supported by recent analyses that adoption of security technologies in the public sector is heavily influenced by assessments of risk versus cost [2]. This implies that lack of widespread adoption of a security technology from the research community in commonly-used software may suggest that the technology in question doesn't meet user needs sufficiently well to justify the cost of implementing it.

We take the more extreme position that the environment in which security technology is deployed is undergoing radical change, and that change is such that current usability problems are only going to get rapidly worse. We argue that attempting to "add on" usability to existing security technology is no more likely to be successful than attempts to "add on" security to existing software systems designed without it, and that new security technologies need to be designed from the ground up with the user foremost in mind.

To design such new technologies, we have embarked on a research program focused around the question: "if you put usability first, how much security can you get?" We approach this question in the context of ubiquitous computing and wireless devices, a context where usability and flexibility are paramount. As part of this program we have begun to design new approaches to integrating security technology into applications, as well as new basic security technologies that enhance usability. We present here several early results of our work and the work of others as examples of new ways to design usable approaches to security.

1.1 Why Now: The Changing Face of Users
The traditional "users" of security technology can be divided into three groups: the developer who integrates security into a system, the administrator who sets policy and monitors se-

curity for a number of users and machines, and the end-user, who must ensure that their own practices follow and help to implement these policies. Research to date has focused on the latter two groups, and in particular on how they use software whose basic function is inherently security-related (see section 2.2).

As computing leaves the desktop and the office, solutions like firewalls and protocols are becoming something that end-users must understand if they are to protect their own devices. Moreover, each individual is more likely to have a multitude of devices, some which will never see a professional system administrator, but all of which must seamlessly work together. In more general terms, end-users are becoming their own systems administrators for increasing numbers of potentially mobile devices [13]. In doing so, users' primary goals are framed in terms of the work they need to accomplish, not in terms of managing security [27, 26], and they are not supportive of technologies that prevent them accomplishing those primary goals [24]. At the same time, attempts to categorize *a priori* in terms of roles or work processes, what they will need to do to accomplish those goals and arrange the necessary privileges to match, have not been successful [18, 8, 23, 24]. If organizational roles and processes could not categorize the office work conducted on desktop computers, then how will an *a priori* categorization succeed in a world where devices are used at work, at home, and in public settings and where the same machine may be used for a multitude of work and leisure activities and sometimes in conjunction with technologies owned by friends and strangers. Indeed, we think it is unlikely that associating security with *a priori* categorizations will adequately support end-users.

Now, more than ever, the challenge to create compelling security is one of making security software *usable* as well as useful, and making software that is not focused directly on security secure. These challenges fall squarely on the shoulders of application developers – in essence, any prescription for designing secure software must focus in large part on them.

1.2 Why Now: The Changing Face of Systems

The dramatic increase in home-based computer networking, the move from desktop computers and centralized servers to laptops, and the proliferation of small wireless devices such as PDAs and cell phones have tremendous implications for computer security, and particularly how it is provisioned to and by end-users. The traditional model of computer security began around controlling access to resources by many users on one central machine whose software served as a potentially trusted base [8]. We are now moving to a world where each user may have many computers or computing devices, of whom s/he is frequently the only user. These devices are overwhelmingly mobile, and may move repeatedly in and out of "managed" environments. Devices may communicate more often with entities outside their managed security infrastructure (if they have one) than entities inside it, but still may need the ability to authenticate and protect those communications. The distinction between "clients" and "servers" blurs – even small wireless devices are servers as well as consumers of information, and need to be protected accordingly. Shared resources that need more

sophisticated access control are accessed over a network on some server that might be in- or outside of the device's administrative domain. At the same time, these machines still need all of the protection mechanisms designed for an earlier, more centralized world – but instead of protecting shared resources from overreaching users on the "inside," they are often protecting a user's machine from actions taken by malicious software or intruders, or even accidental user action.

Devices and systems designed primarily for distributed computing are best served by a different kind of security architecture than their more "inwardly" focused predecessors, for whom distributed computing was an occasional nuisance. First, the fact that they may be often or always out of reach of a managed security infrastructure means that we need tools that allow end-users to effectively manage their own security and that of their devices [15, 20], rather than relying more and more on centralized or outsourced security services. Second, such devices are likely to rely on cryptography more heavily to authenticate their cross-domain interactions [8, 25], and to protect communications on hostile public networks. Traditionally, system designs have tended to minimize or avoid the use of cryptography, even when it was the right tool for the job, because of its perceived slowness and expense [6]. As a result, they have not provided many facilities to ease its use by developers. Though cryptographic libraries are becoming ubiquitous, the infrastructure and operating system components needed to support them (see Section 2.1.2) are not.

1.3 What Now: Don't We Just Need a Better GUI?

There are three conclusions we can draw from the fact that users are unable to make effective use of much of current security technology. First, the problem could be with the users. A really good enough GUI, slightly redesigned applications, or more effective user education about why they ought to choose better passwords could be enough to make them willing and able to tackle security tasks as they are currently formulated [26, 24]. There are indeed cases where usability of password systems can be improved by removing the most egregious demands on end users [3, 11, 24, 22], or adding a nice GUI to a fairly standard piece of security software is enough to more it from the rarefied domain of systems administrators and place it within reach of end-users – the most effective example of this being the recent slew of firewall programs designed for use on end-user machines. However, experimental results have already shown us that this is not the case in general [27] – wrapping a better user interface around existing security technology does not make users suddenly able to effectively use it.

Second, the problem may be with the users – they may simply be completely incapable of understanding or using the technology, and therefore of having any effective control over their own security state. Such risks, and such choices must therefore be taken out of their hands, and security imposed on them uniformly by systems administrators our outsourced services. Here, research has shown us that such an approach is first, not likely to be flexible enough to let users actually perform their intended tasks [8, 18, 23, 26] and second, completely in appropriate to the increasingly common situation where the user *is* the system administra-

tor [13].

The third conclusion is that the problem is primarily with the technology, not the users: even if the technology is good, it is not doing what it was intended to do (make systems more secure *in practice*), and blaming the users will not make it any more effective [24]. The most expedient method to increase the usability of security technologies and applications is to build them from the ground up with usability as their primary focus. Given that now is the time when new security technologies are just beginning to be developed to cope with the world of mobile and wireless devices and ubiquitous computing, now is the time to make sure that those new technologies focus on usability from the outset.

2. HOW DO WE DESIGN FOR USABLE SECURITY?

We present here two of the most pressing issues facing researchers who want to design for usable security: first, how do we look at security from a software engineering perspective, and both design primitives that focus on combining usability and security as well as enable developers to use them effectively. Second, how do we assess how well we've done?

2.1 Three Engineering Approaches to Building Usable Security Technology

2.1.1 Build in Implicit Security

The first, and perhaps most important approach to building usable security technology is to attempt to build what we call *implicit security* into applications – to unify the often "separate but (un)equal" views the user is forced to have of application goals and security operations. Applications which endeavor to be "security-agnostic," and to rely entirely on OS security mechanisms without building in any explicit security knowledge of their own suffer from the following problem: security lives in separate, parallel universe that the user must act on in addition to whatever actions are needed to "directly" accomplish their desired task.

For instance, take the simple task of giving a file to another user. To temporarily share a file using standard file sharing protocols, you must both opt to share the file and change the file permissions so that the destination party (or, if the destination party is not a member of your security infrastructure and cannot be so named, "everybody") can access that file. Afterward, you both have to remember to "turn it all off" – to change your security settings back again. In other words, for each task-oriented action, the user must perform one or more manual steps to manually mirror those application goals in terms of operating system security settings. Contrast that with the steps needed to transfer the document using email. Your access to the document is determined at the point at which you attach it to the email message. Your decision to give the destination party access to the document is indicated by your choosing to attach it to a mail message addressed to them. If you are able to send the message to them encrypted under a key you know to be theirs, you can be reasonably confident that they have actually succeeded in transferring the document to *them*. In other words, your security-related actions can be determined *implicitly* from your goal-directed actions. They do not require separate, error-prone layer of mirroring actions on your part.

We have begun to explore the design of applications that can take advantage of opportunities for *implicit security* – where the user takes an action that indicates both an application goal and a required, *implied* security operation, have these occur automatically as a single step rather than requiring the user to perform multiple parallel operations (see Section 3.3). In such an application, the user performs only actions designed to accomplish application goals, and the application software automatically invokes the required parallel security-related actions.

A simple and effective example of such an application is the Secure Shell (SSH [5]). SSH allows a user to make a tty-based connection to another machine in a fashion almost identical (in terms of user experience) to that used by Telnet. However, the connection made by SSH is encrypted and authenticated. SSH offers fairly sophisticated key management and configuration options, but by default these are not visible to the end-user. In its default configuration, users are authenticated to their target system using standard password-based logins, but those passwords are transmitted over the encrypted tunnel. While the first time a user logs into a target machine, its public key is likely to be accepted based on faith (though the user is given a unobtrusive option to verify the key, and sophisticated users can pre-configure the system with known keys for important target machines), the application automatically tracks the keys of machines the user has connected to, and will warn the user if the target machine presents a different key than the one it has used previously. SSH installation is standard on many (primarily Unix-based) operating systems, and is increasingly effectively set up to auto-configure itself – *e.g.*, to generate and store its long-term machine key pairs automatically the first time it is executed.

While it is not possible to seamlessly integrate security and user goals in every situation, we believe it is a valuable and important technique. Such applications have the advantage of putting the user's goals first and foremost, and attempt to let the user accomplish those goals as directly as possible while at the same time not lowering the actual security of themselves or others in any way not required to accomplish that goal.[1] Applications designed in such a fashion can be, at minimum, drop-in replacements for earlier, less secure versions; at best such applications take advantage of security technologies to enable new activities attractive to end-users.

2.1.2 Refactoring Security Infrastructure

There is a constant tension in the design of technology between which fundamental security mechanisms are part of the operating system and domain infrastructure, and which are left to applications. Facilities provided by the operating system have the advantage that they can be much more strictly enforced, and only have to be written once. Facilities provided by applications must frequently be reinvented anew for every application, and are written by application

---

[1] Also note that even in cases where such an approach would lower the theoretical security of a system, it may increase the system's *effective* security, as users will now use what security measures are in place.

developers who have little interest in and experience with security. Given these criteria, it is usually thought that as little security as possible should be left to applications.

Emphasizing "security-agnostic" applications causes two major problems: first, it is often only the application that has sufficient contextual information to be able to make flexible decisions about access and trust. Second, such applications cannot use the techniques described above to unify user goals and required security operations, and must resort to painful manual mirroring techniques to keep the two worlds in step.

At the same time, there are a number of common infrastructure components that are best handled by the operating system, and an even larger number that will be needed by so many applications that it is simply more effective to write them once and provide them as a service. Standard software engineering techniques such as refactoring [16] can be used to identify such common components. Many such components (e.g., a source of secure random numbers, a relatively secure place to store keying information, a place to store system-wide trust information and a way for domain administrators to update it, etc.) are provided by at least one of the major operating systems in current use, but no single system provides them all. It would greatly ease development of applications that want to use security technology if they could simply rely on every OS to provide such facilities. It would also ease development of applications that would allow domain-based management of some of these components. While such facilities could be provided by a highly portable add-on toolkit instead of incorporating them into the OS itself, such a toolkit would have to be ubiquitous enough that developers could rely on its presence, rather than having to provide it as an extra component to install themselves, be stable and effective enough that there wouldn't be a lot of overhead involved in version compatibility problems, or dealing with incompatible, competing toolkits, and comprehensive enough to provide all of the facilities needed (though this latter problem could be solved by a standard suite of tools). The security-related classes provided by the Java™class libraries, and the combination of the OpenSSL toolkit and /dev/random on Linux begin to achieve this approach for a subset of the functionality we describe here.

2.1.3 Building Lego™Bricks for Security

Finally, we have noticed that well-designed security technologies packaged as reusable components that accomplish a single task effectively are enthusiastically adopted by application developers. A notable case in point is the SSL/TLS protocol [12], as implemented in a number of easily obtainable, fast, and reliable libraries such as OpenSSL and Sun's standard Java™implementation. Developers and designers who are normally not comfortable with security technology or cryptography are comfortable with SSL, almost to extremes – "secure systems" becomes almost synonymous with "systems that use SSL," whether or not SSL is an appropriate tool for the security problems at hand. Kerberos [25] has also provided a frequently reused set of software tools enabling application developers to take advantage of authentication technologies – in this case even to the point of creating a new verb, to "kerberize" a piece of software.

Similarly, technologies we have designed for authenticating secure connections in ad-hoc networks have seen effective reuse (see section 3.2). We believe that application developers can begin to make effective use of very goal-oriented security technologies if provided in forms like these, and have begun actively looking for common security problems in need of such reusable solutions.

Similarly, we believe the field would benefit from creating and using security-related software idioms or "patterns" similar to the software use patterns common in other areas of development [17, 9]. These could help developers less sophisticated in the use of security technology to understand how to incorporate it more effectively into their applications.

2.2 Usability of Security Software is Not (Necessarily) Software Security Usability

Traditional approaches to usability testing focus on observing and interviewing end-users (either in their real-world setting using techniques such as contextual inquiry [7], or in an experimental setting such as a usability lab). The test criteria focus assessing the end-users ability to use the application to get their own or an experimental task done. Success can be measured objectively using metrics such as task completion time and subjectively using interview questions (during the experience or afterward) to find out whether the user found the system easy to use and enjoyed the experience.

A fundamental problem arises when trying to apply these approaches to evaluate the usability of security technology. Usability methods test the usability of end-user applications, and there are very few end-user security applications. Usability testing of security to date has focused on security applications or end-user visible security mechanisms including encryption software [27], password mechanisms [1, 3, 26, 24], and user interfaces for managing policy [28].

In other words, "usability of security" tends to be defined as "usability of security-related applications." While usability of security-related applications continues to be a part of the challenge, we need to broaden and refine our methodological base. This is especially important with distributed systems where every piece of software has to deal with security issues implicitly or explicitly.

We propose three ways of usability testing the usability of secure applications. Each relies on the traditional foundations of data gathering: recording, observing, and interviewing. What we argue here is that we need to reformulate traditional usability testing approaches to accommodate the fact that security is not the primary focus of attention from the end-user perspective, and yet it is end-users' usage that we are most concerned with.

First, we can make use of data logs. Data logs built into the applications can check certain secure system behaviors. For example, when an application sends or receives data logs of what was transmitted can show us whether that data was encrypted. One disadvantage of data logs is that end-users have to be informed about their presence (to ensure fair treatment as subjects in the study). Specifically, we need to provide information about what is being logged, why, how

85

the data will be used, and who will get to access those logs. This can be handled as part of the consent procedure.

Second, we need to design our studies, whether they be experimental or in the field, to ensure that end-users perform tasks that require the use of the security infrastructure. In an experimental approach this requires designing tasks that involve actions and reactions by the end-user that will utilize the security infrastructure. In a contextual inquiry approach where the usability tester observes the end-user doing their normal computing routines at their place of computer use (at work, in a public space, at home) this will require finding appropriate activities that happen "in the wild." Finding naturally occurring "candidate activities" that have security implications can be achieved using fieldwork techniques, such observing and asking questions of potential users about the kinds of work that they are doing and what it involves. After deploying the software the usability tester then visits the end-users when they are most likely to be engaged in those candidate activities and observes whether the software is still helping them achieve their goals.

Third we need to reconsider how and about what we interview individuals about. Asking direct questions about security creates two problems. First, since we have chosen at times to make the security a seamless part of the application (from the end-user perspective) it will be difficult for them to answer questions about things they have not seen or done. Second, asking them questions about security may lead them to change their own sense of security during the interview itself. Despite these difficulties, we can design our interviews to achieve two purposes: ensure that the end-user has a positive "user experience" and focus in on occasions where applications made security decisions visible. By the latter, we mean focus on the times when the security infrastructure requires that the user make choices about what to do and how to do it. We can frame these in terms of the application behavior and other real-world concepts such as privacy.

Finally, in addition to considering the role of human-computer interaction (HCI) for evaluating systems, we also propose considering methods for design. Specifically, fieldwork techniques could also be used to observe candidate settings for secure applications. The results from these kinds of observations and interviews could support the design of end-user applications that leverage the security infrastructure and help it fit (from the end-user perspective) into the activities that are a feature of the setting. In other words, field methods can serve in the requirements part of the software life-cycle as well as the evaluation part.

3. BEGINNING TO GET IT RIGHT: EXAMPLE NEW TECHNOLOGIES AND APPLICATIONS

We present here three examples of technologies and applications that turn security on its head, and look at it from the user's point of view. All three are in active development, and future user testing will tell whether they achieve their goals of putting usable security directly into the hands of the user.

3.1 Identity-Based Encryption

One of the most fundamental problems in all of cryptography is *Key Management* – getting the right keys to the right places at the right times, and being able to trust that you have actually done so correctly. If it is difficult for cryptographers to design in theory, it is well nigh impossible for end-users to handle in practice – and yet the basic operations of key management, namely key pair generation, storage, and public key exchange are those a user must successfully go through in order to begin an application task like exchanging encrypted email [27].

Boneh and Franklin [10] have recently found a practical solution of a long-standing problem in cryptographic research that turns this user problem on its head. An *identity-based encryption* scheme is one where there is a set of (global or domain-specific) parameters shared by all users, and given those parameters, a user's public key can be any arbitrary string – *e.g.*, "beki@parc.com." If you know the system parameters (which you will either get as part of the installation of your mail client, or which your client could retrieve automatically from the DNS server of the intended recipient's domain), you know your intended recipient's public key. You don't need to do any explicit work up front in order to be able to send encrypted email to a particular recipient. When that person receives the encrypted mail, either their mail client already has a copy of their private key and can decrypt it, or they are prompted to perform a one-time retrieval of their private key from the (global or domain-specific) key server, after which they can decrypt new messages automatically. Such a system puts the greatest demands (one-time retrieval of a private key) on the user who gains the most reward (ability to decrypt encrypted email). Such an incentive structure is much more likely to succeed [19] than one that forces the sending user to make a decision every time about whether *this* email is sensitive enough to require obtaining the recipient's public key, or whether they might as well send it unencrypted "just this once."

A demonstration system for identity-based encryption and a plug-in for the Eudora email client program are available on the Internet (http://crypto.stanford.edu/ibe). Work is ongoing in our group and others to take advantage of IBE's usability properties to enable secure networking (IPSEC), and extend its use in email.

3.2 Authentication for Ad Hoc Networks

An increasingly common problem faced by users is that many of the devices and individuals they wish to securely communicate with are not part of their own infrastructure or security domain. Existing security technology concentrates primarily on making authentication and access control decisions about users and activities within that domain, and assumes that no interaction will occur with entities outside the domain without much prior infrastructure arrangement [8, 25]. Cryptographic approaches allow users to exchange data securely with anyone with whom they can exchange an authenticated public key, but current applications don't provide easy ways to let users take advantage of this fact. This means that users have no tools to use to perform secure operations that target or "name" entities outside of their existing security domain, and that non-infrastructure users

(*e.g.*, home users or small organizations) must put in the effort to build an infrastructure before they can secure any of their communications.

A setting where this is particularly important is that of wireless ad-hoc networking: secure communication between arbitrary co-located devices that share no *a priori* trust information other than that their owners wish them to communicate with each other. In particular, we want to capture the user intuition that they want their device to communicate (only) with *that* particular communication partner – to have the act of pointing out who you want to talk to also *implicitly* indicate to the software who was authenticated to communicate with you. The devices in question are typically mobile and communicating primarily over an unsecured wireless network, so are at great risk for man-in-the-middle attacks.

Our solution involves combining sending a small amount of authentication information over a privileged, physically constrained channel, with a key exchange performed over the main wireless link [4]. The authentication information is comprised of a commitment to a public key, and "helper" information like the source's current IP address and desired contact port. This authentication information is transmitted over a channel such as infrared or contact that has the property that it is difficult for an attacker to transmit their own data in that channel without being detected. The protocol is immune to eavesdropping attackers who simply listen in on either the privileged channel or the main wireless link; they don't know the appropriate private keys to be able to impersonate a legitimate conversant.

This approach has many advantages (see [4] for complete details): from the user's point of view, they are simply "pointing out" the printer, laptop, or other device with which they want to exchange information using contact or IR; the system steps in to make sure that they are *only* exchanging information with that desired target device.[2] Any standard, trusted, public-key-based key exchange protocol (*e.g.*, SSL) may be used over the wireless link to secure the bulk of communication. Devices may use single-use ephemeral keys to maintain their privacy, or long-term keys to allow this one authentication event requiring physical proximity to bootstrap secure communication from arbitrary locations in the future. Devices need no pre-existing trust infrastructure or resolvable "names," but can take advantage of them if they have them – *e.g.*, an employee from a corporation, X, that has an established PKI, can use this approach to easily identify the *particular* Corporation X employee he wishes to communicate with now. His software can both ensure that he can only talk to Corporation X employees and use the target's keying information to index into the company database to present additional information about the target (*e.g.*, name, photo) to the user. Additional privileged channel types with *broadcast* capability, *e.g.*, an audio channel, can be used to securely bootstrap group key exchange

---

[2]That device may be malicious, but that is the risk of choosing to talk to strangers. We believe that it should be possible for the user to assume the risk of talking to a particular unknown device without forcing them also to expose themselves and that communication to every other device able to listen to the wireless network.

protocols, for instance in a conference setting.

This simple solution to a constrained problem has become a building block, like SSL, that we find useful in an increasing number of applications. We think that the development of such "usable security" primitives will greatly ease future development of secure applications.

3.3 Application Tasks with Implicit Security

In Section 2.1 we suggested that when possible applications be designed using *implicit security* – when a user takes an action in application terms, s/he also takes a security action – the actions are coupled, and cannot be separated. This reduces the requirements on the user, and helps to prevent the problems of "dangling security state" – where the system's security configuration is not in step with what the user sees, because s/he has forgotten to take one of the explicit mirroring actions necessary to keep the two in sync. Applications that take this approach can much more easily make both application and security state *visible* to the user at all times – such reflection makes it much easier for the user to avoid mistakes and make effective security (and even privacy) decisions [27, 29, 28, 14].

An application we are currently developing [14] takes this approach to letting users share not only files, but services – access to printers, projectors, etc. In this application, users who wish to share things with each other set up a shared "space." Users invited to join the space can see the objects and services in the "space," and know when other users are added to the space. To share a file or a service with the other members of a space, a user simply drops the object onto that space; it then becomes visible to both them and the other space members. This interface conflates visibility with access – if you can see it, you can use it, and if you can't use it, you don't even know it's there. Similarly, the application makes it immediately visible to the user what they are making accessible to others – all data and services that are shared with any space are listed explicitly in a designated panel of the application, making it easy for them to rapidly remove any shared object from all spaces. In some sense, security is primary in this application, as the concept of a "space" is basically an access concept.

At the same time this access information is visible to the user, the details of how it is implemented are not. Security-related tasks happen seamlessly as part of user actions directed toward explicitly application goals. Creating a space causes the creation of root credentials that will be used to secure access to that space. Adding someone to the space involves in part generating credentials for them, that they will later use to prove to other members of the space that they belong to the space. These credentials are created by the user that chose to add them to the space (by that user's software, without their direct involvement). These credentials (essentially public key certificates) allow all members of the space to authenticate other members of the space, even if they were invited in by different people, and map onto the X509 certificates expected by standard SSL implementations, so provide an easy means to encrypt and protect the integrity of all communication between devices involved in the space. Future user testing with this application will make it clear whether we have met our goal of matching

application goals and security actions.

4. CONCLUSIONS

Ubiquitous computing makes usability a critical challenge for security. What has not been clear is the right way to address this problem. We propose a more extreme view than has previously been taken: that underlying security technology must change, and must be redesigned from the beginning with usability in mind. We have provided several examples of cryptographic and security technologies developed using such an approach, and are currently embarking on a research program to develop systems using these technologies and test them with users. We believe that this will require collaboration between the human-computer interaction and security research communities (as is already beginning at this workshop) to ensure that systems are usable and secure. It will be a combination of these interactions that will make security useful for end-users.

5. REFERENCES

[1] A. Adams and M. A. Sasse. Users are not the enemy: Why users compromise computer computer security mechanisms and how to take remedial measures. *Communications of the ACM*, 42:40–46, December 1999.

[2] R. Anderson. Why information security is hard – an economic perspective. In *Proceedings of the 17th Annual Computer Security Applications Conference*, 2001.

[3] R. J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley and Sons, 2001.

[4] D. Balfanz, D. K. Smetters, P. Stewart, and H. C. Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *Proceedings of Network and Distributed System Security Symposium 2002 (NDSS'02)*, San Diego, CA, February 2002.

[5] D. J. Barrett and R. E. Silverman. *SSH The Secure Shell*. O'Reilly, 2001.

[6] T. A. Berson. Cryptographic abundance. *Technology Review*, 105:90–93, 2002.

[7] H. Beyer and K. Holtzblatt. *Contextual Design: A Customer-Centered Approach to Systems Design*. Morgan Kaufmann, San Francisco, CA, 1997.

[8] R. Blakely. The emperor's old armor. In C. Meadows, editor, *New Security Paradigms Workshop*. ACM, 1996.

[9] R. Blakley. Security design patterns. http://www.opengroup.org/security/gsp.htm.

[10] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In *Proc. CRYPTO 01*, pages 213–229. Springer-Verlag, 2001. LNCS 2139.

[11] R. Dhamija and A. Perrig. Dejà vu: A user study using images for authentication. In *Proceedings of the 9th USENIX Security Symposium*, 2000.

[12] T. Dierks and C. Allen. *The TLS Protocol Version 1.0*. IETF - Network Working Group, The Internet Society, January 1999. RFC 2246.

[13] W. K. Edwards and R. E. Grinter. At home with ubiquitous computing: Seven challenges. In *UbiComp '01*, Atlanta, September 2001. Springer-Verlag. LNCS 2201.

[14] W. K. Edwards, M. Newman, T. F. Smith, J. Sedivy, D. Balfanz, D. K. Smetters, H. C. Wong, and S. Izadi. Speakeasy: an extensible framework for peer-to-peer collaboration. In *Proceedings of the Conference on Computer-Supported Cooperative Work*. ACM Press, 2002.

[15] C. M. Ellison. Establishing identity without certification authorities. In *Proceedings of the 6th USENIX Security Symposium*, San Jose, July 1996.

[16] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[18] R. E. Grinter. Supporting articulation work using configuration management systems. *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, 5(4):447–465, 1996.

[19] J. Grudin. Why CSCW applications fail: problems in the design and evaluation of organization of organizational interfaces. In *Proceedings of the Conference on Computer-Supported Cooperative Work*, pages 85–93. ACM Press, 1988.

[20] U. Holmström. User-centered design of security software. In *Human Factors in Telecommunications*, Copenhagen, Denmark, May 1999.

[21] U. Jendricke and D. G. 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*, 2000.

[22] I. Jermyn, A. Mayer, F. Monrose, M. K. Reiter, and A. D. Rubin. The design and analysis of graphical passwords. In *Proceedings of the 8th USENIX Security Symposium*, Washington DC, 1999.

[23] D. Povey. Optimistic security: a new access control paradigm. In *New Security Paradigms Workshop*, Arlington, VA, 1999. ACM.

[24] M. A. Sasse, S. Brostoff, and D. Weirich. Transforming the 'weakest link' – a human/computer interaction approach to usable and effective security. *BT Technology Journal*, 19(3):122–131, July 2001.

[25] J. G. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In USENIX Association, editor, *USENIX Conference Proceedings (Dallas, TX, USA)*, pages 191–202, Berkeley, CA, USA, Winter 1988. USENIX Association.

[26] D. Weirich and M. A. Sasse. Pretty good persuasion:
A first step towards effective password security for the
real world. In *New Security Paradigms Workshop*,
pages 137–143, Cloudcroft, NM, 2001. ACM.

[27] A. Whitten and J. D. Tygar. Why Johnny can't
encrypt: A usability evaluation of PGP 5.0. In
Proceedings of the 8th USENIX Security Symposium,
Washington, DC, August 1999.

[28] M. E. Zurko, R. Simon, and T. Sanfilippo. A
user-centered, modular authorization service built on
an RBAC foundation. In *IEEE Symposium on
Security and Privacy*, pages 57–71, 1999.

[29] M. E. Zurko and R. T. Simon. User-centered security.
In C. Meadows, editor, *New Security Paradigms
Workshop*. ACM, 1996.