Secure Mobile Computing via Public Terminals

Richard Sharp[1], James Scott[1] and Alastair R. Beresford[2]

[1] Intel Research,
15 JJ Thomson Avenue, Cambridge CB3 0FD, UK

[2] Computer Laboratory, University of Cambridge, 15 JJ Thomson Avenue,
Cambridge CB3 0FD, UK

`richard.sharp@intel.com`
`james.w.scott@intel.com`
`alastair.beresford@cl.cam.ac.uk`

Abstract. The rich interaction capabilities of public terminals can make them more convenient to use than small personal devices, such as smart phones. However, the use of public terminals to handle personal data may compromise privacy. We present a system that enables users to access their applications and data securely using a combination of public terminals and a more trusted, personal device. Our system (*i*) provides users with capabilities to censor the public terminal display, so that it does not show private data; (*ii*) filters input events coming from the public terminal, so that maliciously injected keyboard/pointer events do not compromise privacy; and (*iii*) enables users to view personal information and perform data-entry via their personal device. A key feature of our system is that it works with unmodified applications. A prototype implementation of the system has been publicly released for Linux and Windows. The results arising from a pilot usability study based on this implementation are presented.

1 Introduction

It is often convenient to access personal data and applications from public terminals (e.g. viewing documents in a hotel business center, or checking email in an Internet cafe). However, it is also dangerous: public terminals are an easy target for criminals intent on harvesting passwords and other confidential information from legitimate users.

This is not just a theoretical threat. The Anti-Phishing Working Group (APWG) report that the use of *crimeware* (e.g. software-based keyloggers) has recently *"surged markedly"*, with the number of new crimeware applications discovered doubling from April to June 2005 [2]. As well as software-based vulnerabilities, the public terminal hardware itself can be compromised. Tiny, inexpensive devices embedded in keyboards or PS2/USB cables [8] can log millions of key strokes to flash memory. Attackers can easily install such devices on public terminals, leave them for a while and return later to collect users' private data and authentication information. In addition, attackers can often obtain user credentials from public terminals without installing any malicious software or hardware at all. For example, persistent data stored in browser caches sometimes includes usernames/passwords [20] and legitimate *Desktop Search* packages installed on shared terminals enable attackers to browse the private documents and emails of previous users [23]. Of course, when terminals are located in busy public places, attackers can also *shoulder-surf*: stand behind a user in order to read the content of the display and watch them type on the keyboard [25].

In the field of mobile computing research, a great deal of effort has been expended in developing architectures and interaction techniques that exploit shared public terminals. For example, the Internet Suspend Resume [9] project aims to provide users with the ability to *"logically suspend a machine at one Internet site, then travel to some other site and resume . . . work there on another machine"*; the authors of the Virtual Network Computing thin-client technology [18] were motivated by creating a world where *"users can access their personal . . . desktops on whatever computing infrastructure happens to be available—including, for example, public web-browsing terminals in airports"*; and interaction techniques such as *Situated Mobility* [15], *Parasitic Computing* [12] and *Opportunistic Annexing* [16] rely on small mobile devices co-opting computing resources already present in the environment in order to facilitate interaction with their users (e.g. taking over a public display and data-entry device). If mobile computing research is to achieve its goal of providing anytime, anywhere access to personal information via shared public terminals, it is clear that the security and privacy problems highlighted above must be addressed.

In this paper we present a framework that allows users to access their applications and data securely using a combination of public terminals and a trusted personal device such as, for example, a smart phone or a PDA[3]. Our system enables users to enjoy the rich interaction capabilities of large situated displays and keyboards, whilst performing security-critical operations via their smart phone. For example, passwords are typed via the smart phone's keypad, thwarting key-loggers running on the public terminal. Similarly, secure information is displayed only on the smart phone's screen preventing both screengrabbing attacks perpetrated by software running on the public terminal[4]. Using the smart phone for private display and data-entry also helps protect against shoulder-surfing. A major benefit of our approach is that we do not require applications to be re-written; our system is specifically designed to work with existing Windows, Linux and MacOS applications.

[3] Although a variety of small personal devices with displays and keypads are adequate for our purposes, for the sake of brevity, the remainder of this paper will refer to the trusted personal device as a smart phone. In Section 2 we will justify our belief that smart phones are indeed more trusted than public terminals.

[4] Packages such as *PC SpyCam 2.0* combine key logging and screen grabbing to build a complete picture of what a user did with a computer. Programs such as this can often be configured to take screenshots in response to specific actions (e.g. when a user opens their email package).

The key contributions of this work are: (*i*) A general system architecture suitable for the scenario above (Section 2), (*ii*) A *threat model* formalising the attacks against users of public terminals, and a set of *security principles* addressing these threats (Section 3), (*iii*) a *GPL implementation* of the architecture (Section 4), which is equally applicable to both the *Remote Access Model* [18] where the user's thin-client server is located on a remote machine accessible via the Internet; and the *Personal Server Model* [26], where the user's applications run locally on their personal device, and (*iv*) a *pilot usability study* showing the feasibility of our approach (Section 5). We also survey related research (Section 6) and present conclusions and directions for future work (Section 7).

2 System Overview

Our system is based on thin-client technology. When a mobile session is initiated, the smart phone and the public terminal connect concurrently to a user's thin-client server as shown in Figure 1. On connection, the public terminal's display shows the user's entire desktop; the phone's (smaller) display shows a scrollable portion of this screen. As well as performing data-entry via the public terminal, the user can also type via the keypad (or virtual keypad) on their smart phone.

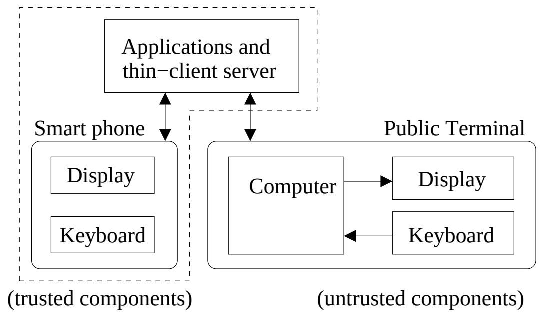
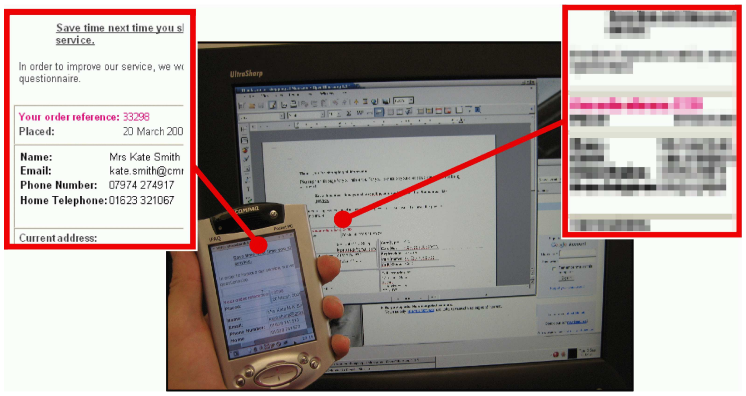


Fig. 1. Using our system to access private information via a public terminal.

When performing operations that are not security critical the user interacts with the public terminal in the usual fashion, via its own display and keyboard. However, when a user performs a security critical operation (e.g. entering a credit card number) they can use their smart phone to activate a variety of security features. The security features currently supported include (*i*) applying image processing filters to censor content on the public display; (*ii*) controlling the way in which (untrusted) mouse and keyboard events arising from the public terminal are interpreted; and (*iii*) entering security-critical data, such as passwords, via the (trusted) smart phone's keypad.

The security provided by our system is based on the premise that a user's smart phone is inherently more trustworthy than a public terminal (e.g. it is less likely that crimeware will be running on a user's smart phone than on a public terminal). There are a number of reasons why this assumption is justified: (*i*) whereas it is easy for hackers to gain physical access to a public terminal (in order to install crimeware, for example), it is much harder to gain physical access to a users' phone; (*ii*) users install applications on their phones relatively infrequently, and often only inside sandboxes such as Java MIDP which do not permit general keylogging/screengrabbing, thus limiting the risk of trojan-based crimeware[5]; and (*iii*) the developers of phone-based operating systems often go to great lengths to prevent installed applications from performing silent network communication—this makes it difficult for crimeware to transmit information (such as keylogs etc.) back to hackers without alerting the user. (We note that, in previous work, other security researchers have made similar arguments, claiming that personal devices offer a greater degree of security than general purpose PCs [3].)


Save time next time you sl
service.

In order to improve our service, we wo
questionnaire.

Your order reference: 33298
Placed: 20 March 200

Name: Mrs Kate Smith
Email: kate.smith@cmr
Phone Number: 07974 274917
Home Telephone: 01623 321067

Current address:

Fig. 2. Our prototype in use. Content on the public terminal is censored (right inset); the area around the mouse pointer appears uncensored on the private display (left inset). In this case, pixellation is used for censoring the public display, but other filters are also provided.

Figure 2 shows a picture of our system in the case where a user has opted to censor the content on the public display. When content on the public display is censored, the area of the screen surrounding the mouse pointer[6] is automatically displayed *uncensored* on the smart phone's private display. As the user moves the mouse pointer, the uncensored smart phone's display scrolls accordingly. Although one cannot use the censored public display to read the text of the private document, it still provides a great deal of contextual information—for example, the positions of windows, scroll bars, icons etc. Users can thus perform

[5] Legitimate applications which incorporate crimeware functionality.

[6] We assume that the public terminal provides some kind of pointing device: e.g. it is a touchscreen, or a mouse or tracker-ball is available.

macroscopic operations (e.g. dragging or scrolling windows) via the censored public display, whilst performing *microscopic operations* (e.g. selecting options from a menu or reading a private email) via their smart phone's private display.

3 Security Model

Following standard security engineering practice we start by presenting our *threat model* and *security policy model* [1]. The threat model characterises attackers' motivation and capabilities; the security policy model provides an abstract, architecturally-independent description of the security properties of our system.

3.1 Threat Model

Attackers' motivation is to steal private and confidential information, often with a view to committing identity theft and fraud. Attackers are capable of mounting both *passive monitoring attacks* and *active injection attacks* against the public terminal. Passive monitoring attacks include recording everything shown on the public terminal's display, typed on the public terminal's keyboard and transmitted over the network. Active injection attacks include injecting malicious data packets into the network and also injecting fake User Interface (UI) events (e.g. keypresses and mouse clicks) into the public terminal.

For an example of an attack based on injecting fake UI events, consider the following. An attacker installs crimeware on the public terminal that waits until the user opens their email client. At this point the crimeware generates click events, selecting each email in turn. When the emails are displayed on the screen (as a result of the injected click events) the attacker performs a screen grab, thus obtaining users' private information.

We assume that users' smart phones are not compromised and that attackers therefore have no means of either recording or injecting phone-based keyboard, screen or UI events. However attackers can nonetheless monitor and inject network packets travelling to and originating from the phone.

3.2 Security Policy Model

We address the threat model presented above by adopting the following four security principles:

1. *The connections between the smart phone and the application server must be authenticated and encrypted.* This protects against network monitoring and injection attacks[7].

[7] In the case where it is more probable that the network is compromised than the public terminal itself, one may also consider authenticating and encrypting the connection between the public terminal and the application server.

2. *Users must be able to enter text via their (trusted) phone at all times in the interaction.* This protects against keylogging attacks since information entered via the phone cannot be recorded by keyloggers running on the public terminal.
3. *Users must have control over what is shown on the public display* (e.g. show everything, remove all text, turn off entirely). This protects against malicious screengrabbing software running on the public terminal.
4. *Users must have control over how events originating from the public terminal are interpreted.* For example, users may tell applications to ignore all keypresses from the public terminal, allowing only keypresses originating from their phone; similarly, users may instruct applications to ignore all mouse clicks originating from the public terminal. This protects against User Interface injection attacks.

4 Technical Details

Our system is based on the well-known, open-source VNC thin-client software [18]. Figure 3 presents a diagrammatic view of our system's architecture, showing the dataflow of control messages and UI events between the thin-client server, public terminal and smart phone. (Although we start by assuming that the thin-client server is running on a separate machine, Section 4.3 shows that this general architecture is also applicable to the Personal Server Model [26] where the user's applications run locally on their smart phone.) Like VNC, we use the Remote FrameBuffer (RFB) protocol [24] to send pixel data from the server to remote displays and to send mouse and keyboard events from remote input devices back to the server. However, we have modified both VNC server and client software in order to enforce our Security Policy Model (above). Our source code is freely available for download [22].

As shown in Figure 3, the VNC server is augmented with 3 extra components: the *Image Processor*, *Authentication Manager* and *UI Security Controller*. These components are described below.

Image Processor: This component is responsible for removing private content from the public display, censoring the image transmitted to the public terminal as requested by the user. We have currently implemented three censoring algorithms: *uniform blur*, *pixellation* and *text removal*. (Our image processing algorithms are discussed in more detail in Section 4.1.)

Authentication Manager: In accordance with our Security Policy Model, all data transmitted between the thin-client server and the smart phone is tunnelled over the standard SSH protocol [27]. SSH already contains provision for two-way client/server authentication. The public terminal is authenticated by means of a one-time password displayed on the smart phone's screen. The authentication manager relies on SSH's standard authentication primitives to determine which

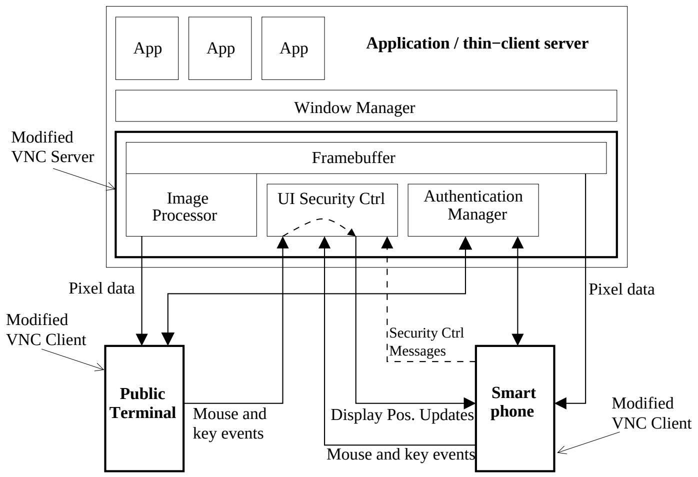


Fig. 3. Dataflow of control messages and UI events through the thin-client server, public terminal and smart phone.

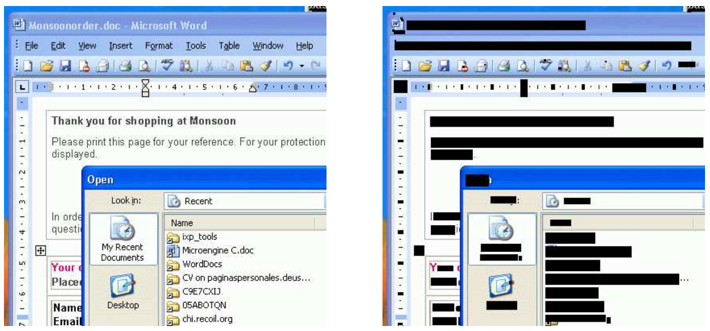
of the connections to the thin-client server originates from the trusted smart phone and which originates from the public terminal, configuring the UI Security Controller and Image Processor components accordingly.

UI Security Controller: The UI Security Controller filters mouse and keyboard events generated by the public terminal in order to protect against UI injection attacks; (this functionality will be discussed in detail in Section 4.2). The UI Security Controller also processes *Security Control Messages* generated in response to a user activating or deactivating a particular security feature via their smart phone. As shown in Figure 3, the UI Security Controller forwards mouse movement events originating from the public terminal to the smart phone. These display position updates are interpreted by the smart phone as an instruction to move its uncensored display window to a new screen location, as described in Section 2.

4.1 Image Processing Components

The Image Processor maintains a second framebuffer containing a censored version of the screen. This framebuffer is the one exported to the public terminal's display. To avoid constantly recensoring the entire framebuffer, the Image Processor is hooked into VNC's screen update events; only screen areas that have changed are reprocessed. The Image Processor also understands VNC's *Copy-Rect* events [24]—when areas of the screen that have already been censored are copied or moved (e.g. when the user drags a window) they are not reprocessed.

We have implemented three image processing filters that users can activate or deactivate via their smart phone: pixellation, blurring and text removal. The


Monsoonorder.doc - Microsoft Word
File Edit View Insert Format Tools Table Window Help
Open
Look in:
Recent
Name
My Recent Documents
Desktop
ixp_tools
Microengine C.doc
WordDocs
CV on paginaspersonales.deus...
C9E7CXIJ
05ABOTQN
chi.recoil.org

Thank you for shopping at Monsoon

Please print this page for your reference. For your protection displayed.

Fig. 4. The text detection and removal filter. *left*: a portion of the uncensored public terminal screen; *right*: the same portion after text removal has been activated.

pixellation and blurring filters are both parameterisable—the user can set the level of pixellation or blurring using their smart phone. It is worth noting, however, that whatever level of blurring or pixellation is used, attackers may be able to use sophisticated *image restoration* software to reconstruct some of the censored information. To protect against this eventuality, in the case where a user wants to hide text from attackers, we implemented a text removal filter. This component explicitly detects areas of text, replacing them with filled rectangles.

The text removal filter is based around a 5-stage image processing pipeline: (*i*) transformation to a 1-bit (binary) image by means of adaptive thresholding; (*ii*) building a tree of contours using edge detection on the 1-bit image; (*iii*) marking possible text by contour analysis; (*iv*) applying heuristics to remove false positives; and (*v*) blanking out regions which are believed to contain text. The full technical details of the image processing are beyond the scope of this paper; interested readers may wish to examine the source code for more information [22]. Our approach is similar to recent work on general text and information extraction from images and video [7].

We tested our text removal filter on a variety of Windows applications, including Word, Excel, Internet Explorer, Adobe Reader and Visual Studio; a typical example is shown in Figure 4. In general we found that text removal was near 100% effective. However, performance dropped to a 92% success rate on Adobe Reader due to its heavily anti-aliased fonts. Most of the detection failures on Adobe Reader only revealed single characters, although in one case an entire word went undetected. We measured the speed of the algorithm on an Intel Pentium 4 (3.2 GHz) machine, by executing the algorithm on a variety of typical Windows screenshots (at a resolution of 1280x1024). The average processing speed was 102 fps (*s.d.* 4 fps).

In future work we intend to improve the text removal filter to handle anti-aliased text better. Nonetheless, our current implementation demonstrates that text removal through image processing is a viable method for censoring a public display, in terms of both accuracy and execution time.

4.2 Dealing with untrusted mouse/keyboard events

To protect against UI injection attacks (as described in Section 3.1) untrusted mouse and keyboard events originating from the public terminal are filtered by the UI Security Controller. In this section we describe the various filtering policies supported. Users can activate or deactivate mouse/keyboard filtering policies at any time using their smart phone.

We currently support 3 different policies for dealing with keyboard events from the public terminal: ignore all keyboard events entirely, allow all keyboard events from the public terminal and only allow alphanumeric/cursor keypress events. This latter policy deserves further explanation—we argue that it provides a convenient tradeoff between allowing the user to perform many common tasks (e.g. editing a document) whilst making it more difficult for an attacker to mount dangerous key-injection attacks (which often rely on using Alt- or Ctrl- key combinations to access application functionality).

Of course, only allowing alphanumeric key events does not preclude key-injection attacks entirely—some applications may allow critical functions to be accessed via alphanumeric key presses; furthermore, when a user clicks on the "File" menu in a Windows application, for example, alphanumeric and cursor keys can be used to select and execute critical functions (such as "Open New", "Exit" etc.). However, filtering all but alphanumeric keypress events nonetheless makes it *more difficult* to execute key-injection attacks, offering users a sweet-spot on the security-usability spectrum.

Pointer events from the public terminal (mouse, touchscreen, tracker ball etc.) are also filtered by the UI Security Controller. We provide two mouse filtering modes: one which allows all mouse events from the public terminal to pass through unmodified (in the case where the user is not concerned about mouse event-injection attacks); and one which allows only *mouse movement events* from the public terminal, but filters all click events. The second of these policies is based on the principle that it is difficult to mount a dangerous UI-injection attack by inserting mouse movements alone. Of course, it is also difficult for the user to perform any legitimate actions without clicking, so to address this we map one (or more) of the smart phone's buttons to click events; in essence these buttons on the smart phone take the place of the regular mouse buttons. This mode of operation gives the user the best of both worlds—they can use the public terminal mouse (or touchscreen or trackerball etc.) to point at arbitrary screen areas; however, click events are generated by the phone and thus trusted.

One *can* envisage scenarios in which dangerous mouse event injection attacks may be executed entirely via faking mouse movement events. However, just as filtering all but alphanumeric keypress events makes key-injection attacks harder, our model of removing clicks from the public terminal's event stream makes mouse-injection attacks harder. Again, we feel that this mode of operation finds a sweet-spot on the security-usability spectrum.

4.3 Adapting our Architecture to the Personal Server Model

Figure 3 shows three distinct devices all communicating over the network: the user's thin-client server (which may, for example, be located in their office, at home or in an ISP data-center), the user's smart phone and the public terminal. This is the *Remote Access Model* as envisaged, for example, by the creators of the X Window System [21], VNC [18] and the Internet Suspend/Resume project [9]. However, our architecture is equally applicable to the *Personal Server Model* [26], in which users' personal data and applications are not accessed over the Internet, but instead reside locally on their personal mobile device—in our case, on their smart phone.

We can map our architecture onto a Personal Server Model by assuming that the user's smart phone runs not only our thin-client viewer, but also the thin-client server, window manager and applications. Whilst the architecture as presented in Figure 3 remains broadly the same, adopting the Personal Server Model results in two significant differences to the system as a whole. Firstly, the job of the Authentication Manager becomes simpler—the trusted client is the one connecting via the loopback interface (since both trusted client and thin-client server now reside on the smart phone). Secondly, there is no longer any need for Internet connectivity. The smart phone (containing all applications and personal data) can simply connect *directly* to the public terminal via a communication technology of choice (e.g. Bluetooth, WiFi or even a USB cable).

We observe that modern smart phones are already powerful enough to support a thin-client server and general purpose applications. The Motorola E680, for example, contains a 500 MHz Intel XScale Processor. As the trend of increasing computational power on mobile devices continues we believe that smart phones running both applications and thin-client servers have the potential to form the basis of a powerful mobile computing platform.

5 Pilot Usability Study

We performed a pilot usability study that primarily aimed to address one key question: can novice users interact with their personal applications and data via our system, even when all the security measures are activated? We chose to focus particularly on this question because it is one of the most fundamental; after all, if participants find the combination of a smart phone display and censored public display too difficult to interact with, our architecture offers little value.

We simulated a scenario in which participants accessed emails and documents using a combination of a large censored display and a small, mobile uncensored display. We used a 19-inch flat panel monitor and a mouse/keyboard to simulate the public terminal, and an iPaq to simulate the participant's smart phone. We installed our system (as described in Section 4) on the public terminal and iPaq and configured it as shown in Figure 2: as participants moved the mouse, the pointer moved across the censored public display and the iPaq's display scrolled to show the uncensored screen area surrounding the pointer. As described in

Section 4.2, buttons on the mouse were disabled—to click participants used one of the four buttons on the iPaq's physical keypad. The keyboard on the public terminal was also disabled. The study was performed using the Mozilla web-browser and OpenOffice Writer applications running over the Gnome Desktop on Linux. We used a pixellation filter to censor content on the public display (see Section 4.1). The level of pixellation was sufficient to render all text used in the study completely illegible.

8 participants took part in the study: 4 male, 4 female. The average age was 29 (*s.d.* 5.7; min 25, max 39). Participants were an educated, mostly professional group; 4 participants worked in computing, 4 came from non-technical backgrounds. All used computers on a regular basis and all owned mobile phones. No participants reported any difficulties with vision or motor skills; all were right handed.

Participants performed two tasks. *Task A* involved using the Mozilla web-browser to access a Gmail account containing 8 emails. Participants were presented with the Gmail Inbox page and asked four comprehension-style questions (e.g. "What did people buy Bob for his birthday?"). To answer the questions participants had to navigate the Gmail interface and read the content of emails—some of the emails were in the Inbox and some were in the Sent Mail folder. *Task B* involved using OpenOffice Writer, a word processing package similar to Microsoft Word. Participants were presented with the OpenOffice Writer application and asked to open 3 documents in turn: an order receipt from a large UK-based clothing company (1 page long), a personal CV (3 pages long) and a product order form for educational materials (2 pages long). The process of opening a document involved selecting *File→Open* from the menu bar or clicking on the Open File Toolbar Icon, and then navigating a File Browser Dialog Window. For each document, participants were asked two comprehension-style questions, requiring them to find specific pieces of information. Since the documents were too large to fit on the public terminal, participants had to scroll the OpenOffice Writer window down in order to find the answers to some of these questions.

Before starting the tasks participants were allowed to practice using the system for as long as they liked. In this familiarisation phase, participants were given the free reign of the Gnome Desktop to play with. (All participants finished practicing within five minutes.) To avoid ordering effects half the participants performed Task A first, and half performed Task B first. We used the *think-aloud protocol* [10] and conducted the studies under quiet, office conditions with only the participant and one researcher present. During the study, we recorded the time taken for the participant to complete each task; afterwards we performed structured interviews in order collect qualitative data regarding the participant's experiences of the system.

Our key result was that all participants were able to complete the tasks without prompting and in a reasonable time. Task A was completed in an average of 82 seconds (*s.d.* 16s; min 56s, max 195s); Task B was completed in an average of 196 seconds (*s.d.* 42s; min 124s, max 252s). Participants answered all comprehension questions correctly. This gives an unequivocally positive answer to our initial question: novice users can interact with applications and data via our system, even with all the security measures activated (i.e. disabled public keyboard, disabled mouse buttons and censored public display).

We were interested in finding out whether participants felt that the censored public display was useful, or whether they preferred to rely on the iPaq's screen alone. All participants claimed that they found the censored public display useful for Task B, where they used it to scroll through different pages of documents and navigate File Dialog windows. One participant commented *"I couldn't use ... [the censored public display] to read anything, so I used it more as a map I suppose. Without it I would have found it hard to find scroll bars, menus and open files."*. However, for Task A, only four of the participants claimed that the censored public display was useful. For example, one participant observed *"in the GMail task I only used the big screen a little bit to find the back button on the browser; everything else I just did through the [iPaq's] little screen."*. We believe that the reason participants relied less on the public display in Task A was that this task did not require participants to scroll or navigate between multiple windows; it is for these kinds of actions that the *"map"* provided by the censored public terminal is particularly useful.

Six participants commented that they initially found it confusing to click with the iPaq buttons rather than with the mouse buttons. A typical response was *"at first I kept clicking with the mouse button, which didn't do anything, but as it went on I started to get the hang of clicking with the other [iPaq] button. It's not difficult though, you just get used to it."* The fact that participants consistently made comments about confusion relating to clicking with the iPaq buttons suggests that users would probably prefer to leave the mouse buttons enabled whenever possible. At present we feel that leaving mouse buttons enabled would not present a great security threat; of all the attacks highlighted in our threat model (see Section 3.1), mouse injection attacks are probably the most unlikely in practice. One participant made an interesting suggestion: *"when you accidentally click with the mouse button the system could maybe beep to remind you to click with the iPaq button."* The idea of incorporating automatic, interactive help features into the system that assist the user in this kind of way is something we would like to explore in future work.

We see this study as providing promising initial results. However, we recognise that there are many other important questions that we have not yet addressed. For example, would users be able to understand *when* they should activate/deactivate security measures (e.g. when to type on their smart phone, when to censor content on the public display)? Are users sufficiently concerned about security to use our system at all when using a public terminal? These are questions that we intend to address in further studies. However, as a precursor to studying the second question, we note that a number of participants explicitly mentioned that they were concerned about shoulder-surfing when accessing personal information in public spaces. For example, *"Being able to blur the [public]* *screen is really useful. I don't want people to see what I'm reading—I'm really scared of people looking over my shoulder."*

6 Related Work

We have already highlighted how our research relates to number of other projects including VNC [18], The X Window System [21], Internet Suspend/Resume [9] and the Personal Server [26]. In this section we further explore the relationship between our work and the research of others.

A number of researchers have proposed blurring electronic information in order to provide privacy. However, these projects have tended to focus on protecting users' privacy in always-on video used to support distributed workgroups [5, 28]. In contrast we focus on general purpose access to applications and data, addressing not just privacy concerns (e.g. screen blurring) but also other ways in which public terminals may be attacked in order to violate privacy, including keylogging and UI-injection attacks.

Berger *et al.* developed an email application which blurred sensitive words on a projected display; selecting blurred words caused them to appear (uncensored) on a private wrist-watch display [4]. A calendaring application with similar functionality was also proposed. The concept of blurring words on a public screen, whilst allowing them to be displayed on a personal private display is similar to ours. However, our work extends this in two ways: firstly we present a security model and architecture that enables users to access *existing*, unmodified applications securely via public terminals; secondly, as well as *displaying* private content, we also deal with secure mouse and keyboard *input* (to avoid logging and UI-injection attacks), issues not considered by Berger *et al*.

Ross *et al.* developed a web-proxy which split an HTML page into secure content and insecure content according to a user-specified, programmable policy [19]. The secure content was then displayed on a WAP browser running on a mobile device whilst insecure content was displayed on a public display. Again, the idea of splitting content between personal/public devices in this way is similar to ours. The relationship of our work to Ross' is similar to the relationship of our work to Berger's research: (*i*) our framework is immediately applicable to *all* applications (not just web browsing); and (*ii*) as well as dealing with displaying sensitive information on public displays, we also consider secure *input* to applications.

Our thin-client architecture for securing mobile computing systems is similar to that of Oprea *et al.* [14]. The major difference between this project and our work is that Oprea does not provide mechanisms for obfuscating the content on the untrusted display whilst viewing portions of it in unobfuscated form on a trusted personal device. Indeed Oprea states that *"it turned out that the performance of the RFB protocol and VNC software on our PDA was too poor to make this approach work efficiently"* [14]. In contrast we have shown that it is possible to implement this interaction technique efficiently on top of VNC/RFB[8] and, further, that novice users can cope with dual displays without significant difficulty.

The idea of simultaneously using multiple displays to access applications and data has been explored extensively by the research community [11, 17]. Our work adopts these ideas, simultaneously using users' smart phone display and public terminals to enable secure access to personal information via situated displays and input devices.

Many researchers have explored users' privacy concerns surrounding accessing information via public displays [13]. Our system addresses these concerns, enabling users to access documents and applications via situated displays while allowing them to view sensitive information privately via their personal phone display.

7 Conclusions and Future Work

In this paper we have presented a thin-client architecture capable of supporting secure, mobile access to unmodified applications. Our system allows users to benefit from the rich interaction capabilities of large situated displays, whilst relying on a trusted personal device to protect them against the inherent insecurity of shared, public terminals.

The implementation described in this paper gives users full control over which security policies to apply in which contexts (e.g. when to type on the phone keypad rather than the public keyboard, when to censor content on the public display etc.). This is fine for experts, but further studies are required to determine whether non-technical users can successfully select the right policies to apply in order to protect their privacy. In future work we would like to explore whether automated *activity inference* methods may benefit novice users, automatically suggesting suitable security settings in different contexts. Another possibility is to explore ways that enable service providers (such as banks, for example) to specify a particular security policy, removing some of the control from users. In this scenario, web services could be explicitly written to enforce a pre-determined split between a general purpose PC and a trusted, personal device (e.g. credit card numbers and account transfers are *always* performed via an interface on the personal device, whereas statements may be browsed on the public terminal).

While this paper discusses a thin-client implementation of our Security Policy Model (Section 3.2), other implementations are also possible. For example, we may choose to implement a system which works at the window-manager level, enabling users to (say) select which windows to censor on the public display and which to leave uncensored. Similarly, we could implement a system for secure web-browsing on public terminals which works at the HTML-level, using an HTTP proxy to censor parts of the web page (c.f. [6]) and to migrate secure input fields and hyperlinks to the personal device.

[8] At least for the personal-server model where applications run on the mobile device, (see Section 4.3) or remotely when a low-latency network connection is available.

Our thin-client approach has the advantage of working with existing applications, with the disadvantage of having a coarser granularity of privacy controls than the other solutions above. However, we can also achieve the "best of both worlds"—by exposing an API from the UI Security Controller, applications themselves can specify detailed security policies for particular areas of the display. Thus, the user is afforded always-present basic privacy controls at the framebuffer layer, while also enjoying the usability advantages of precise privacy-control support in whatever suitably enabled applications they may have.

Acknowledgements

The authors would like to thank Tim Kindberg, Claudio Pinhanez and Ian Smith for insightful comments on a draft of this paper.

References

1. Ross Anderson, Frank Stajano, and Jong-Hyeon Lee. Security policies. In *Advances in Computers vol 55*. Academic Press, 2001.
2. Anti-Phishing Working Group (APWG). Phishing activity trends report, June 2005. `http://antiphishing.org/`.
3. Dirk Balfanz and Ed Felton. Hand-held computers can be better smart cards. In *Proceedings of USENIX Security*, 1999.
4. S. Berger, R. Kjeldsen, C. Narayanaswami, C. Pinhanez, M. Podlaseck, and M. Raghunath. Using symbiotic displays to view sensitive information in public. In *Proceedings of PERCOM*. IEEE, 2005.
5. Michael Boyle, Christopher Edwards, and Saul Greenberg. The effects of filtered video on awareness and privacy. In *Proceedings of ACM CSCW*, 2000.
6. Richard Han, Veronique Perret, and Mahmoud Naghshineh. WebSplitter: a unified XML framework for multi-device collaborative web browsing. In *Proceedings of CSCW 2000*. ACM, 2000.
7. Keechul Jung, Kwang In Kim, and Anil K. Jain. Text information extraction in images and video: a survey. *Pattern Recognition*, 37:977–997, 2004.
8. Amecisco KeyLogger product range. `http://www.keylogger.com/`.
9. M. Kozuch and M. Satyanarayanan. Internet suspend/resume. In *Proceedings of the WMCSA 2002*, June 2002.
10. C. Lewis and J. Rieman. Task-centered user interface design—a practical introduction, 1993. University of Colorado, Boulder. (This shareware book is available at `ftp.cs.colorado.edu`).
11. Brad A. Myers. Using handhelds and PCs together. *Communications of the ACM*, 44(11):34–41, 2001.
12. Chandra Narayanaswami, M. T. Raghunath, Noboru Kamijoh, and Tadonobu Inoue. What would you do with 100 MIPS on your wrist? Technical Report RC 22057 (98634), IBM Research, January 2001.
13. Kenton O'Hara, Mark Perry, and Elizabeth Churchill. *Public and Situated Displays: Social and Interactional Aspects of Shared Display Technologies*. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

14. Alina Oprea, Dirk Balfanz, Glenn Durfee, and Diana Smetters. Securing a remote terminal application with a mobile trusted device. In *Proceedings of ACSA 2004*. Available from `http://www.acsa-admin.org/`.
15. Trevor Pering and Michael Kozuch. Situated mobility: Using situated displays to support mobile activities. In *Public and Situated Displays: Social and Interactional Aspects of Shared Display Technologies*. Kluwer, 2003.
16. Jeffrey S. Pierce and Heather Mahaney. Opportunistic annexing for handheld devices: Opportunities and challenges. In *Proceedings of HCIC 2004*, 2004.
17. Mandayam Raghunath, Chandra Narayanaswami, and Claudio Pinhanez. Fostering a symbiotic handheld environment. *Computer*, 36(9):56–65, 2003.
18. Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, 1998.
19. Steven J. Ross, Jason L. Hill, Michael Y. Chen, Anthony D. Joseph, David E. Culler, and Eric A. Brewer. A composable framework for secure multi-modal access to Internet services from post-PC devices. *Mob. Netw. Appl.*, 7(5), 2002.
20. Ken Salchow. Sorting through the hype of ubiquitous secure remote access and SSL VPNs. SecurityDocs white paper. `http://www.securitydocs.com/library/3103`.
21. Robert W. Scheifler and Jim Gettys. The X window system. *ACM Trans. Graph.*, 5(2):79–109, 1986.
22. Richard Sharp, James Scott, and Alastair Beresford. Resources and code accompanying this paper. `http://www.cambridge.intel-research.net/securemobilecomputing/`.
23. Tom Spring. Google Desktop Search: Security Threat? Today@PCWorld. `http://blogs.pcworld.com/staffblog/archives/000264.html`.
24. T. Richardson, RealVNC Ltd. The RFB Protocol, 2005. `http://www.realvnc.com/docs/rfbproto.pdf`.
25. Desney S. Tan and Mary Czerwinski. Information Voyeurism: Social impact of physically large displays on information privacy. In *Extended Abstracts of CHI 2003*. ACM, 2003.
26. Roy Want, Trevor Pering, Gunner Danneels, Muthu Kumar, Murali Sundar, and John Light. The personal server: Changing the way we think about ubiquitous computing. In *Proceedings of UbiComp 2002*. Springer-Verlag, 2002.
27. T. Ylonen. SSH transport layer protocol. RFC 3667.
28. Qiang Alex Zhao and John T. Stasko. The awareness-privacy tradeoff in video supported informal awareness: A study of image-filtering based techniques. Technical Report 98-16, Georgia Institute of Technology, 1998.