Fighting Crimeware: An Architecture for Split-Trust Web Applications

Richard Sharp, Anil Madhavapeddy, Roy Want, Trevor Pering, John Light

IRC-TR-06-053

Research at Intel

Fighting Crimeware: An Architecture for Split-Trust Web Applications

Richard Sharp
Intel Research
15 JJ Thomson Avenue
Cambridge, CB3 0FD. UK
richard.sharp@intel.com

Anil Madhavapeddy
Computer Laboratory,
University of Cambridge,
15 JJ Thomson Avenue
Cambridge, CB3 0FD. UK
avsm2@cam.ac.uk

Roy Want
Intel Corporation
2200 Mission College Blvd
Santa Clara, CA 95054
roy.want@intel.com

Trevor Pering
Intel Corporation
2200 Mission College Blvd
Santa Clara, CA 95054
trevor.pering@intel.com

John Light
Intel Corporation
2200 Mission College Blvd
Santa Clara, CA 95054
john.light@intel.com

Abstract

We present an architecture for split-trust browsing: a technique that enables web applications to split their HTML across a pair of browsers—one untrusted browser running on a PC and one trusted browser running on a user's personal device. Information entered via the personal device's keypad cannot be read by the PC, thwarting PC-based keyloggers. Similarly, information displayed on the personal device's screen is also hidden from the PC, preserving the confidentiality and integrity of security-critical data even in the presence of screengrabbing attacks and compromised PC browsers. We present a Security Policy Model for split-trust web applications that affords defence against a range of crimeware-based attacks, including those based on *active-injection* (e.g. inserting malicious packets into the network or spoofing user-input events). Performance results of a prototype split-trust implementation are presented, using a commercially available cell phone as a trusted personal device.

1 Introduction

As people are increasingly relying on the web for security critical tasks, *crimeware*—malicious software designed expressly to facilitate illegal activity—is being used to steal identities and commit fraud. The Anti-Phishing Working Group (APWG), a global consortium of companies and financial institutions focused on eliminating Internet fraud, report that the use of crimeware has recently "*surged markedly*" with the number of new crimeware applications discovered doubling from April to June 2005 [6]. The trend is so marked that the APWG believe that ultimately "*conventional phishing via social engineering schemes will be eclipsed by advanced, automated crimeware*" [1].

To date the most prevalent form of crimeware is the *keylogger*: a program that secretly records users' keypresses, transmitting sensitive information (e.g. credit card numbers, usernames and passwords) back to criminals. Other examples of crimeware include applications that record the contents of users' screens, silently redirect web browsers to attackers' websites and maliciously spoof user-input to control web applications (e.g. trigger a money transfer in an on-line bank) [29, 17].

Technically savvy individuals have always been wary of the threat of crimeware on public terminals (e.g. Internet cafes). Worryingly, however, the recent wave of crimeware attacks has involved malicious

applications installing themselves on users' personal PCs, either as trojans or by exploiting OS-level vulnerabilities [18].

The threat of crimeware poses fundamental challenges to the web's security model. In particular, although HTTPS/SSL protects data as it is transmitted between client and server, it cannot protect data from compromised end-points. For example, as soon as the contents of an HTTPS URL has been decrypted by the Secure Socket Layer it can be snooped by trojan browser-extensions, screengrabbers and other forms of crimeware. Similarly, HTTPS/SSL does not preserve the privacy or integrity of user input; malicious applications running on the PC can, for example, record key presses and even fake user input (e.g. generate a spoofed click event on a hyperlink).

Our research addresses the threat of crimeware by allowing people to browse the web using a combination of a general-purpose networked PC and a personal, more trusted device. For the most part, a user browses the web via the PC as normal. However, security-critical operations are performed via their personal device, using its display and keypad for I/O. Information entered via the personal device's keypad cannot be read by the PC, thwarting PC-based keyloggers. Similarly, information displayed on the personal device's screen is also hidden from the PC, preserving the confidentiality of security-critical data even in the presence of screengrabbing attacks and compromised PC browsers. We believe that combining general purpose PCs with trusted personal devices to provide a unified browsing platform gives users the best of both worlds: they can enjoy the rich browsing capabilities of their PC, with its large display and full-sized keyboard *and* the greater degree of trust associated with viewing/entering security-sensitive data via their personal device.

The question of what constitutes a *trusted* personal device is an interesting one, and one which is currently topical within the mobile computing industry [4, 11]. One could imagine manufacturing a small, locked-down device with the specific purpose of augmenting a user's web browsing to provide enhanced security. Alternatively, one may argue that *some* existing cell phones or PDAs already provide a more secure computing platform than general purpose PCs and can thus be used as trusted personal devices to some extent. Further discussion of what constitutes a trusted personal device will be deferred until Section 6.1.

1.1 Research Contribution

The technical contribution of this paper is an architecture for *split-trust web browsing*: a technique that enables web applications to split their HTML across a pair of browsers—one untrusted browser running on a PC and one trusted browser running on a user's personal device. As well as splitting content across the PC and personal device, our architecture also allows HTML Forms to be split. In this way secure fields (e.g. credit card details) can be filled in on the trusted personal device, while fields that do not contain sensitive information (e.g. delivery dates or product selections) can be filled in on the PC. In addition to exploring the systems issues surrounding split-trust web-browsing, we also present a Security Policy Model for split-trust web-applications and consider a range of attacks against split-trust systems in general.

1.2 System Overview

Figure 1 shows a high-level overview of our architecture for split-trust browsing. The (untrusted) PC connects to the web server over the Internet, using HTTP to request web pages in the usual manner. The trusted personal device connects *directly* to the PC using a suitable data-link technology (e.g. USB, Bluetooth, WiFi).

The HTML fetched from the web server contains both regular content, which is rendered in the PC's browser in the usual way, and encrypted messages destined for the trusted personal device[1]. The *Remote Device Communication (RDC) Agent*, which runs on the PC, is responsible for forwarding such messages

[1]For the purposes of this paper we assume that web applications are written explicitly to support split-trust browsing.

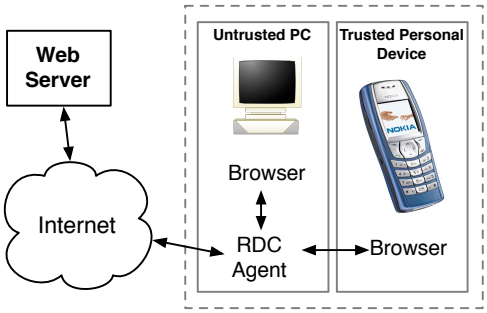

Figure 1: High-level overview of our system for split-trust browsing

Figure 2: (Left) Browsing on the PC while entering security-critical information via the cell phone; (Right) A close-up of the phone screen

between the web server and the personal device. When a message is received by the personal device it is decrypted and displayed on its screen. Similarly, messages generated by the personal device (as a result of user input) are encrypted before being sent back to the web server via the RDC Agent. The session key used to encrypt these messages is known only to the trusted personal device and the web server; crimeware running on the untrusted PC is thus unable to read the encrypted web content.

A critical feature of our architecture is that it does not require the personal device to establish a separate Internet connection to the web server. Instead we tunnel data sent between the web server and the personal device over the PC's existing Internet connection, relying on the RDC Agent to demultiplex these two logical channels. We believe that this model offers a number of benefits over the "two separate Internet connections" approach: (*i*) it provides a better user-experience, since the low-latency direct connection between the personal device and PC means that the two devices can be kept in tight synchronisation with each other; (*ii*) it does not require the user to incur the extra cost of a separate Internet connection for their personal device—e.g. over GPRS or 3G; (*iii*) it means that our architecture is applicable to personal devices that do not support Internet connectivity but still provide direct, point-to-point data connections—e.g. a PDA with a USB link; and (*iv*) it enables tight integration with client-side functionality such as tabbed browsing—when the user clicks on a different browser tab on the PC, the RDC Agent traps this event and updates the screen of the user's personal device accordingly.

Figure 2 shows our system being used to make a secure e-commerce transaction, using a Motorola E680 cell phone as a trusted personal device. The PC browser is used for non-security-critical tasks: browsing the product catalogue, making selections etc. However, when the user starts to purchase the goods, the form requesting credit card details automatically appears on their cell phone. The user fills in these private

details via their cell phone's keypad and selects "submit" from their phone to make the purchase. Crimeware running on the PC is not able to read the content displayed on the phone; nor is it able to snoop the user's keypresses to steal their credit card details.

Although, for the sake of simplicity, this paper assumes that web applications have been written explicitly to support split-trust browsing, the architecture described could be layered on top of existing applications via HTML-rewriting proxies. The design of such proxies and mechanisms for specifying the required transformations is a topic of future work.

1.3 Structure of the Paper

We start by formalising crimeware-based attacks and presenting a set of general design principles that enable split-trust web applications to address these attacks (Section 2). Technical details of our split-trust browsing implementation are then presented (Section 3), followed by a case-study of a split-trust banking web service (Section 4). Various attacks against split-trust web-applications are considered with discussion of how well our architecture defends against each of them (Section 5). Finally, after describing related work (Section 6), we conclude and present directions for future work (Section 7).

2 Security Model

In this section we present our *threat model* and *security policy model* [5].

2.1 Threat Model

Attackers' motivation is to steal private and confidential information, often with a view to committing identity theft and fraud. We assume that attackers are capable of using crimeware to mount both *passive monitoring attacks* and *active injection attacks* against the PC. Passive monitoring attacks include recording everything shown on the PC's display, typed on the PC's keyboard and transmitted over the network. Active injection attacks include injecting malicious data packets into the network, injecting malicious data packets into the direct connection to the personal device and also injecting fake User Interface (UI) events into the PC (e.g. spoofing a click on a hyperlink, or spoofing keypresses to fill-in and submit an HTML form). Further, we assume that the PC-based browser is untrustworthy. For example, crimeware running on the PC may cause the browser to silently redirect the user to an attacker's web site, or to maliciously generate/rewrite HTML (e.g. modify link/form targets, add/remove content).

We assume that the user's personal device is free of crimeware and that attackers therefore have no means of either recording the contents of its screen or data entered via its keypad. HTML received via the PC is rendered faithfully in the personal device's browser, and user-input performed via the personal device's keypad is relayed correctly back to the PC.

2.2 Security Policy Model

As outlined in Section 1.2 we address the threat model presented above by migrating security-sensitive parts of the interface to a trusted personal device. However, in order to benefit from the security provided by this browsing model, a split-trust web application must satisfy the following five properties:

1. *The end-to-end communication channel between the web server and the trusted personal device must be authenticated and encrypted*. This prevents an attacker from snooping traffic between the web server and the phone. It also prevents an attacker from maliciously injecting fresh data into this channel.

2. *All security-sensitive form fields must be filled in via the trusted personal device*. Combined with Property 1, this prevents the untrusted PC from snooping any security-sensitive data entered by the user.

3. *All security-sensitive information must be displayed only on the trusted personal device*. Combined with Property 1, this prevents the untrusted PC from snooping any security-sensitive information served by the web application.

4. *The web application must not allow form submissions from the trusted device to be replayed*. This prevents an attacker from maliciously re-using previous security-sensitive form data entered on the personal device in subsequent transactions.

5. *All security-critical operations must be initiated (or confirmed) via a form on the trusted personal device. Further, there must be sufficient information displayed on the personal device's screen to specify fully the action being initiated*. Combined with Properties 1 and 4 this ensures that crimeware on the untrusted PC cannot subversively initiate an unauthorised security-critical operation (e.g. a money transfer in an on-line bank) without alerting the user.

Properties 1 to 3 are self-explanatory; however, Properties 4 and 5 require further elaboration. We will consider these properties in reverse order, starting with Property 5.

The first part of Property 5 is straightforward: security-critical operations must be initiated or confirmed via the trusted personal device. The motivation for this is clear—by forcing security-critical operations to be confirmed on the trusted personal device, the untrusted PC cannot subversively initiate such operations without alerting the user.

The second part of Property 5 is more subtle and protects against a class of attacks highlighted by Balfanz and Felton [7]. To understand its purpose, it is first helpful to consider the following analogy. Unscrupulous Charlie arrives at Bob's office and says "please sign the following authorisation to transfer $100 from your bank account to Alice's bank account". However, while saying this, he hands Bob a piece of paper which says only "I authorise the money transfer". Bob signs the paper and Charlie takes it to the bank. As he passes it to the cashier he says "here's the authorisation to transfer all funds from Bob's bank account to my bank account". The cashier checks Bob's signature and performs this transfer. The security flaw here is obvious: the authorisation slip is not specific enough; as a result Charlie is able to fool Bob into believing it means one thing, whilst fooling the bank that it means something else.

Unless web applications specify confirmation dialogues for security-critical operations carefully, there is a direct analog of this attack that can be played out in a split-trust browsing scenario. Consider the following example. An on-line bank's web server generates an HTML page which is rendered on the untrusted PC's browser and contains two links: one with text "click here to transfer $100 to Alice's bank account", and one with text "click here to transfer all funds to Charlie's bank account". The browser on the untrusted PC has been subverted so that it maliciously swaps the link targets over: the link with text "transfer $100 to Alice's bank account" now points to the action of transferring all funds to Charlie's bank account and vice-versa. The user clicks on one of the links and, in accordance with the first part of Property 5, a confirmation form appears on the screen of their trusted personal device asking them to authorise the money transfer. It is now clear why the text of the confirmation must *"specify fully the action being initiated"*. If the confirmation is under-specified—e.g. if the text reads only "please confirm money transfer"—then the user is not alerted to the attacker's ploy of swapping the link targets. However, if the confirmation is specified fully—e.g. the text reads "please confirm the transfer of all funds from your account to Charlie's account"—then the user is immediately alerted to the fact that the action currently being performed is not the action they thought they had initiated. The user thus decides not to confirm the action and no money is transferred.

We now turn our attention to Property 4, which specifies that a web application must not allow form submission messages from the trusted personal device to be replayed (i.e. a web application must not accept

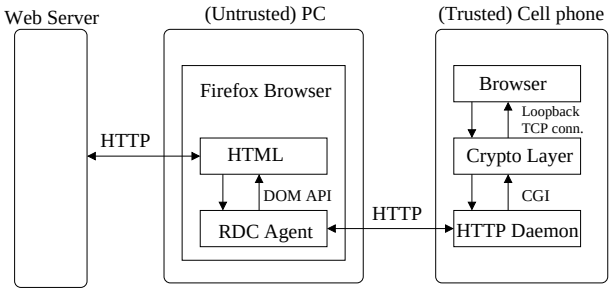

Figure 3: Architecture diagram showing components running on both the untrusted PC and the trusted personal device

data arising from the same form submission action more than once). To see why this is important, consider the following attack. An on-line banking system sends a form to a user's trusted personal device asking them to confirm a money transfer to Alice's account. When the user submits the form (via their trusted personal device), the (untrusted) PC records the resulting submit message. Although an attacker cannot read the contents of this message (since Property 1 requires that it is encrypted with a key known only to the personal device and the web server), they can nonetheless *replay* it in response to a subsequent transaction. Thus, an attacker may maliciously initiate another money transfer to Alice's account (e.g. by spoofing a click-event on the "transfer money" link in the untrusted PC's browser) and then replay the user's previous confirmation message in order to complete the transfer.

Without Property 4 an attacker could thus circumvent our requirement that users explicitly confirm every security-critical operation. This is why the explanatory (non-italic) text of Property 5 observes that it is only when "combined with Property 4" that it ensures "crimeware on the untrusted PC [is prevented from] initiating any unauthorised security-critical operations".

3 Technical Details

We have built a prototype split-trust browsing framework using a commercially available cell phone (Motorola E680) as a trusted personal device. In this section we present the technical details of our implementation.

Figure 3 shows the main components of the system. We run the Firefox browser on the untrusted PC; our RDC Agent is implemented as a Firefox Browser Extension [9]. The cell phone runs a simple cHTML [16] browser which has been implemented as a Java MIDlet.

On initiating a split-trust browsing session, a user connects their cell phone to the PC using a local communication technology of choice—e.g. USB, Bluetooth, WiFi. They execute our extended Firefox browser on the PC and start surfing. As usual, regular (non-split-trust) web sites appear entirely on the PC. However, if the user visits a web-application that supports split-trust, then security-sensitive parts of its interface automatically appear on their cell phone.

The HTML fetched from a split-trust web application contains (*i*) regular content, rendered on the PC as usual; and (*ii*) a number of AES-encrypted [10], Base64-encoded [15] embedded messages. Each of these messages contains cHTML [16] content that may ultimately appear on the personal device's screen. The RDC Agent, running inside Firefox, extracts embedded messages from the received HTML and forwards them to the phone over HTTP (see Figure 3).

The cell phone runs a local HTTP Daemon that receives an HTTP Request from the RDC Agent and, via CGI scripts, passes the embedded message contained within it to the Crypto Layer. There it is decrypted

```
<html ...> <head>
  <title>Split-Trust Browsing Example</title>
  <meta name="split-trust-browsing" content=""> </head>

<body> <!-- This HTML will be rendered on the PC browser as usual:>
<h2>Click on a link below to display secure message on trusted personal device:</h2>

<p><a name="rdc-onClick-0" class="personaldevice" href="JavaScript::">Link 1</a>
<p><a name="rdc-onClick-1" class="personaldevice" href="JavaScript::">Link 2</a>

<!-- --------------- Messages for the personal device embedded here: ------------------- -->
<form name="rdc-data">

  <!-- Default content, displayed on personal device when page loaded:>
  <input type="hidden" name="rdc-onLoad-msg" value="oYW5rcyBmb3IgY2xpY2tpb ... nZS4=====">

  <!-- This message is displayed on personal device when user clicks on link 'rdc-onClick-0' -->
  <input type="hidden" name="rdc-onClick-0-msg" value="WW91ciBWUE4gYWNjb3VudCBk ... a9gfI======">

  <!-- This message is displayed on personal device when user clicks on link 'rdc-onClick-1' -->
  <input type="hidden" name="rdc-onClick-1-msg" value="IGxvZ2luIGRldGFpbHMgYXJl ... VFNDQ=====">
</form>
```

Figure 4: An example HTML page containing embedded messages for the trusted personal device.

before being rendered in the phone's browser. The Crypto Layer is also responsible for encrypting the contents of form fields filled-in on the cell phone before this data is sent back to the RDC Agent on the PC. To simplify user-interface issues the phone's browser does not allow hyperlinks; instead, all hyperlinks reside on the PC-side interface.

Since communication between the PC and cell phone is performed at the HTTP layer our architecture is agnostic regarding the data-link technology connecting the PC and cell phone; as long as an IP-level connection to the cell phone is available our system works. We have tested our implementation with the phone connected over both Bluetooth and USB. For Bluetooth we relied on the standard PAN Profile [2] to provide IP-level connectivity; for USB we used a Windows Device Driver on the PC which provides an IP-level Virtual Network Interface over USB.

In the remainder of this section we describe the architectural components outlined above in more detail. We start by showing how messages for the personal device are embedded into regular HTML pages (Section 3.1); we then describe the implementation of the RDC Agent (Section 3.2) and briefly outline the design of the components running on the cell phone (Section 3.4). For simplicity, our initial description of the system does not consider the splitting of HTML forms. The details of how form fields can be split between the PC and personal device are described separately (Section 3.5). Finally, we present a performance evaluation of our implementation (Section 3.6).

3.1 Embedding Split-Trust in HTML

Figure 4 shows an example HTML page that may be served by a split-trust-enabled web application. A single `meta` tag with attribute `name="split-trust-browsing"` specifies that this page contains embedded messages destined for a trusted personal device. By examining the contents of form `rdc-data` one can see that the page contains 3 such embedded messages, each stored in the `value` attribute of a hidden field. On loading the page Firefox renders the HTML in the usual way, displaying the `<h2>` and the two `<a>` tags on the PC's screen. (Since the messages for the personal device are embedded in hidden form fields they do not affect the page layout.)

The `name` attribute of a message's enclosing form field specifies the event that the message is associated with. For example, in the page shown in Figure 4, the message contained within the field entitled

rdc-onLoad-msg is forwarded to the personal device as soon as the browser has finished loading the HTML. Names prefixed "rdc-onClick" are reserved for messages triggered by click events. In Figure 4 the message contained in the field entitled rdc-onClick-0-msg is associated with the link defined by the <a> tag with name rdc-onClick-0. Similarly, message rdc-onClick-1-msg is associated with link rdc-onClick-1. When the user clicks on a link, the RDC agent checks if there is an associated message and, if there is, forwards it to the trusted personal device. Although not shown in Figure 4, other names refer to different types of events. For example, we could have named a link rdc-onMouseOver-3 and provided a corresponding message entitled rdc-onMouseOver-3-msg.

3.2 RDC Agent

We implemented the RDC Agent as a Firefox Browser extension, writing it in a combination of JavaScript and XUL [9]. Whenever a page is loaded[2] the RDC Agent first checks to see if the split-trust-browsing meta tag is present (see above). If this is not found the RDC Agent stops processing immediately, ensuring that the extension does not degrade the performance of non-split-trust sites. If the meta tag is present, the Browser extension uses the DOM API [13] to check if there are any <a> tags prefixed rdc-. For each of these <a> tags an event listener is added with a callback function that forwards its associated message to the personal device. Finally, if there is a form field named rdc-onLoad-msg then the message it contains is forwarded to the personal device immediately.

Messages are forwarded to the personal device over HTTP; the RDC Agent uses the XMLHttp API [3] to make HTTP Requests without changing the HTML displayed in the browser. The RDC Agent constructs an XMLHttp object containing the message to forward as a POST parameter and a URL prefixed with the IP address of the personal device[3].

3.3 Authentication and Key Exchange

A prerequisite to transmitting encrypted messages between the web server and the personal device is the negotiation of a session key between these two parties. Several existing Internet standards define secure key-exchange mechanisms, such as SSHv2 (rfc4253) [31], IKE (rfc2409) [14] and SSL/TLS (rfc2246) [12]. Our current implementation uses SSHv2 authentication/key-exchange, specifically diffie-hellman-group1-sha1[4] with RSA host keys. We did not use the SSHv2 Diffie-Hellman Group Exchange mechanism due to the additional round-trip of packets required, but this can easily be added for increased security if desired. The RDC Agent acts as an intermediary for the authentication/key-exchange process.

A split-trust web application initiates key-exchange and authentication by serving an HTML page containing a meta tag with name="kex-init". The RDC agent detects the presence of this tag and sends an HTTP Request to the personal device requesting its first key exchange message—M_1 in Figure 5. The RDC Agent receives M_1, contained in the body of the HTTP Response, and copies it into a new HTTP Request which is sent to the web server. The web server responds with its key exchange reply, M_2, which the RDC Agent forwards to the personal device via another HTTP Request etc. Thus, by making alternate HTTP Requests between the personal device and the web server, the RDC agent co-ordinates the flow of cryptographic messages necessary for key exchange (the dotted lines of Figure 5). We do not describe the structure or contents of M_1 and M_2 in detail here since the diffie-hellman-group1-sha1 protocol is well documented in rfc4253 [31]. However, we briefly note that authentication and protection against

[2]Note that since encrypted messages are embedded at the HTML-layer, the RDC Agent works regardless of whether Firefox fetches pages from the web application over HTTP or HTTPS.

[3]The IP address of the personal device is associated with a network interface of whose name is known a priori (e.g. "BNEP0" for Bluetooth, "USB0" for USB). Thus, we can automatically find the IP address of the personal device by using a System API call (e.g. a WinSock call) to request the IP currently assigned to network interfaces of interest.

[4]Which, despite its name, actually specifies the use of Oakley *Group 2* DH Parameters!

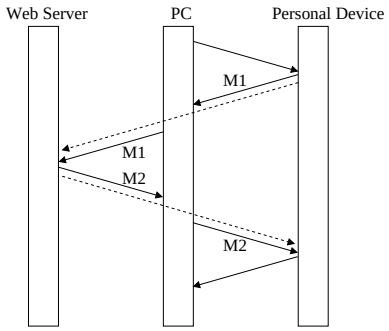

Figure 5: Using the RDC Agent to negotiate a key exchange between the web server and personal device over HTTP RPC calls

man-in-the-middle attacks is provided by a hash, H, of various protocol elements signed with the server's private host key and included in M_2. On receiving M_2 the client recomputes H for itself, verifies the server's public host-key by means of a certificate and then checks the server's signature on H.

When the phone has authenticated the web server (verifying its host-key by means of a certificate) it displays a confirmation dialogue on its screen informing the user of the web server's identity and asking if they want to proceed. Thus, if crimeware on the PC has silently redirected the browser to an attackers site, this fact will be revealed to the user via their trusted personal device. (Redirection attacks will be considered more deeply in Section 5.1).

The `value` attribute of the `kex-init` meta-tag contains a *continuation URL* akin to a `form`'s `action` attribute. When the key exchange/authentication protocol has been completed, the RDC Agent redirects the browser to this URL. In this way a web application can request a key exchange and then, once a session key, S_k, has been established, redirect the browser to show a new split-trust page in which embedded messages are encrypted with S_k. Note that key exchange is not limited to the start of a split-trust browsing session—the web server can request a new session key at any time by means of a `kex-init` meta-tag.

3.4 Components on Cell Phone

We implemented a prototype Crypto Layer for the cell phone (see Figure 3). The multi-precision modular exponentiation required for the key-exchange/authentication protocol relies on the open source *GNU Multi-Precision Arithmetic library* (libGMP), which we cross-compiled for the phone. An open-source AES reference implementation was also cross-compiled for the phone in order to decrypt messages received from the RDC Agent and encrypt phone-based user input.

For technical reasons we were unable to interface our system with the phone's built-in browser; instead, we implemented a simple cHTML browser as a Java MIDP Application in order to display content on the cell phone. The Java browser interfaces with the (native) Crypto Layer via a loopback TCP connection. The implementation of the phone's browser is made considerably easier by the fact that hyperlinks are not permitted on the personal device (see Section 3).

3.5 Dealing with Forms

So far we have seen how a split-trust web application can embed encrypted content in HTML pages, and how the RDC Agent running on the PC can forward this content to be displayed on the cell phone when specific events occur. Here we show how this framework can be extended to deal with split HTML forms in which some fields are displayed on the PC while others appear (and are filled in) on the cell phone.

As with regular content, forms to be displayed on the phone are encrypted and embedded in the HTML messages served by the split-trust web application. For example:

```
<a name="rdc-onClick-0" ...>
  Click here to enter credit card details</a>
...
<form name="myForm" action="..." method="POST">
<field type="hidden" name="rdc-onClick-0-msg"
   value="AKHJ3VAORTU49 ... LGHUBVEBJ1084XZ0===">
<field type="hidden"
   name="rdc-onClick-0-response" value="">
```

When the user clicks on the <a> tag named `rdc-onClick-0` (on their PC) the RDC Agent forwards `rdc-onClick-0-msg` to the personal device in the usual manner. This message can contain a mix of cHTML content and form fields which are rendered in the phone's browser. If, after decrypting a message, the phone finds that it contains form fields, it relays this information back to the RDC-Agent in its HTTP Response (see Figure 3). This triggers the RDC-Agent to poll the phone for the user's response (via repeated HTTP Requests).

The user fills in the form fields via their phone's keypad and selects "Submit" in their phone's browser. The Crypto Layer, running on the phone, encrypts this user input and returns it to the RDC-Agent in an HTTP Response. When an encrypted response is received, the RDC Agent inserts it into the `value` attribute of field `rdc-onClick-0-response` (see above). Thus when `myForm` is submitted, the web application receives data entered on the cell phone via the contents of this field.

Of course, the untrusted PC may maliciously put the encrypted messages in the wrong form field's `value` attribute before submission. To protect against this attack the encrypted message generated by the personal device actually contains a set of ($\langle$*fieldname*$\rangle$, $\langle$*user-input*$\rangle$) pairs. On receipt of a form input message from the trusted personal device the web application parses both the fieldname and corresponding user input ensuring that, even if messages are swapped by the untrusted PC, the right user input is bound to the right field.

A single form can contain fields displayed on both the PC and the phone. In the above example, `myForm` could contain regular (i.e. not hidden) fields which would be rendered by the Firefox Browser in the usual way. On submitting the form, the web application thus receives the values of those fields entered on the PC, as well as encrypted form response messages from the personal device.

3.5.1 Form Submission

There are two alternative mechanisms of submitting split-trust form data back to the web server. Firstly, an application can specify that a form should be submitted by means of a "submit" button displayed on the PC's browser. This is achieved by simply adding a regular `submit` button to the HTML above.

Secondly, an application can instruct the RDC Agent to submit a form automatically as soon as a response is received from the phone. In the above example the web application can request this behaviour by including:

```
<field type="hidden" name="rdc-onClick-0-submittype" value="automatic">
```

Automatic submission is ideal for scenarios such as phone-based login: as soon as a username and password are entered and confirmed on the phone's keypad the web-application proceeds to the next page. In contrast, manual submission (via a button on the PC's browser) is often suitable for pages that contain multiple phone-based forms. In this case users can fill in each of the forms on their cell phone before finally clicking submit in the PC's browser to transmit all this data back to the web application.

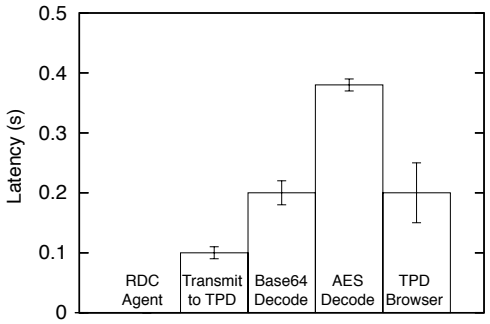

Figure 6: Latencies of the individual components of our implementation (averaged over 20 trials). Error bars show standard deviations. TPD abbreviates Trusted Personal Device—in this case, the Motorola E680 smart phone.

For each phone form, a web-application can also include a corresponding *status* element, displayed on the PC (e.g. `<p name="rdc-onClick-0-status">`). When the RDC Agent forwards a form specification to the phone it simultaneously updates the `innerHTML` property [13] of the corresponding status element (rendered on the PC) to inform the user that the form is "currently being edited on the phone". Similarly, when a user response is received, the status element is updated to notify the user that a "form submission has been received from the phone".

3.5.2 Avoiding Replay Attacks

Recall that Property 4 of our Security Policy Model (Section 2.2) requires that form data entered via the phone must not be subject to replay attacks. To enforce this property we require that each encrypted form specification served by the web application contains a fresh *nonce* [26] and a timestamp. The phone's browser automatically copies this information into its encrypted form response message. On receiving a form response message the web application decrypts it and then checks (*i*) that it has not seen the nonce before; and (*ii*) that the response is timely.

3.6 Performance Evaluation

To assess the performance of our implementation we measured the latency incurred between a user performing an action (e.g. clicking on a link) and an associated 850 byte message appearing on the phone's screen. The message is encrypted using AES with a 1024-bit key and Base64 encoded; our choice of 850 bytes is very much worst case—we expect most messages sent to the phone to be significantly smaller than this.

Our PC was a 2.5GHz Pentium 4 with 512Mb RAM; our trusted personal device was a Motorola E680 smart phone, which has a 400MHz Intel XScale (Bulverde) Processor and 32MB RAM / 32MB Flash. Each of the measurements were averaged over 20 trials. As shown in Figure 6, the latency of each of the components of the system is as follows:

1. The time taken between the RDC Agent receiving a UI-event and initiating an HTTP Request containing the message to be forwarded is negligible (invariably less than 1 ms).

2. With the phone connected to the PC via USB, the time taken to send the HTTP Request containing the encrypted 850 byte message to the phone is 0.1s (*s.d.* 0.01s).

3. The time taken to Base64 decode the message on the phone is 0.2s (*s.d.* 0.02s).

4. The time taken to AES-decrypt the message on the phone, using a 1024-bit key, is 0.38s (*s.d.* 0.01s).

5. The time taken to send the decrypted message to the Java Browser (over a loopback TCP connection) and to render the content on the phone's screen is 0.2s (*s.d.* 0.05s).

Thus the average end-to-end latency between the user generating an event on the PC (e.g. clicking on a link) and the corresponding 850 bytes of content being rendered on the phone's screen is 0.88s. Even for this worst-case message size we believe that 0.88s falls within the limits of acceptable latency for web usage models (since it is comparable to the time taken to fetch a page from a web server over the Internet). Since the time complexity of Base64 decoding and AES decryption is $O(n)$, the latency would reduce linearly with message size.

Regarding the performance of key exchange, using libGMP the Motorola E680 is able to generate a 1024-bit random number and compute a modular exponentiation using Oakley Group 2 Diffie-Hellman Parameters [14] in an average of 0.06s (*s.d.* 0.004s). Thus the time taken to perform key exchange and authentication is most likely to be dominated by the round-trip-times of the HTTP messages initiated by the RDC-Agent (see Figure 3).

4 Case Study

As a case study we consider a split-trust banking application. We start by describing the details of the login process (Section 4.1) and then briefly explore some of the design alternatives surrounding two further aspects of the application: executing money transfers (Section 4.2) and viewing statements (Section 4.3).

4.1 Logging In

When a user clicks the "login link" (on their PC) the server responds with an HTML page containing a `kex-init` meta tag (see Section 3.3). This triggers the RDC Agent to initiate a key exchange as shown in Figure 5. When key exchange is complete, the server records the newly constructed session key in its backend database[5] and, on the client-side, the RDC-agent redirects the PC browser to the main login page, the URL of which has been placed in the `kex-init` tag's `value` attribute. (Recall that, as part of the key exchange and authentication process, the identity of the service provider will be disseminated to the user via their trusted personal device.)

The main login page contains (*i*) some regular HTML (rendered on the PC), informing users that they should use their personal device to enter their username and password; and (*ii*) an encrypted login form (specifying username and password fields) for the trusted personal device. When generating this page, the web server encrypts the login form for the personal device dynamically, retrieving the session key under which to perform the encryption from its backend database[6].

The encrypted login form specification is stored in an `rdc-onLoad-msg` field, instructing the RDC Agent to forward this message to the personal device straight away. The server also includes an `rdc-onLoad-submit` field with the `value` attribute set to `"automatic"`; this specifies that data is to be returned to the web application as soon as the user selects submit on their phone (see Section 3.5).

In accordance with Policy 5 of our Security Policy Model (see Section 2.2) the login form, displayed on the trusted personal device, must contain sufficient text to describe fully the action that is about to be performed (e.g. "enter your username and password to login to BigBank's personal banking service").

After the user has logged in the PC browser displays the main banking interface page; users can click on the links in this page to view statements, perform money transfers, check direct debits etc.

[5] As usual, a unique cookie stored on the PC could provide a *session identifier* used to index into the backend database's session table.

[6] Recall that the webserver also includes a fresh nonce and a timestamp in the encrypted form specification—see Section 3.5.

4.2 Executing a Money Transfer

A money transfer is clearly a security critical operation and thus, to comply with Policy 5 of our Security Policy Model, it must be initiated or confirmed on the trusted personal device (see Section 2.2). However, there are a number of other design tradeoffs regarding the PC/personal device split that depend on the precise details of the bank's and user's security concerns.

For example, if it were decided that attackers must be prevented even from acquiring detailed *knowledge* of a transfer (i.e. account numbers and amounts), then the form that collects this information from the user must appear on the trusted personal device. Alternatively, if it were decided that the only real threat is attackers initiating bogus money transfers, then the usability of the application could potentially be enhanced by displaying this form on the PC. Of course, a fully specified confirmation dialogue must still subsequently appear on the trusted personal device. The important difference is that, in this latter case, the user is able to enjoy the PC's mouse, keyboard and large display whilst specifying the details of the transfer.

4.3 Viewing Statements

Depending on individual privacy concerns and the context in which the on-line banking application is being accessed, users may have different preferences regarding how the statement viewing interface is split between their PC and personal device. For example, a user viewing statements on a public terminal in an Internet cafe may prefer to have their statements displayed on their personal device. Conversely, a user viewing statements at home may have enough confidence in their PC to want to display their statements directly on their PC's screen.

To deal with these different requirements, the banking application could provide a per-session configuration interface allowing users to specify whether statements are to be displayed on their PC or on their personal device. When generating HTML the web server would look up the configuration settings for the current session in its backend database and use this information to determine the PC/personal device split dynamically.

Of course, an act of downgrading security—in this case, changing from a mode in which statements are viewed on the personal device to one in which statements are viewed on the PC—must itself be considered a security critical operation and hence, in accordance with our Security Policy Model, be confirmed via the trusted device. (Otherwise PC-based crimeware could always obtain statements by first spoofing a request to change the configuration to display statements on the PC, and then spoofing a request to display statements.)

5 Attacks Against Split-Trust Browsing

In this section we consider a number of attacks against split-trust browsing and consider how well we can defend against them.

5.1 Phishing

Crimeware attacks are different from conventional phishing attacks: whereas the former rely on malicious software running on users' machines (e.g. keyloggers), the latter rely entirely on social engineering, attempting to fool users into unwittingly entering security-sensitive information into attackers' websites. This paper has motivated split-trust browsing primarily as a technique for addressing PC-based crimeware attacks. However, the general split-trust browsing technique can also be leveraged to address conventional phishing. For example, the server may validate the identity of the user by means of challenge/response authentication with their personal device (cf. one-time passwords). Alternatively, we may combine split-trust browsing with a *password hashing* [24] scheme. In this case, a password entered on the personal trusted

device is hashed with some known properties of the website (including its domain name) before being sent back to the server[7]. Both these techniques would make it harder for phishers to obtain reusable credentials.

Another possible phishing-style attack involves redirecting the untrusted PC to a similar-looking domain name and then presenting a valid certificate for the fake domain. Although, at session-initiation time, a message would appear on the trusted personal device asking if the connection should proceed, the user may not spot that the company/domain name is incorrect. They may therefore click continue and unwittingly connect to the attacker's server.

This is a general problem with certificate-based authentication that we do not claim to have solved. However, as a side note, we observe that we can leverage users' mobile devices to make *physical* certificate exchange practical. For example, we may forbid the trusted personal device from accepting any certificates over the network. Instead, users may present their trusted personal devices at trusted retail outlets and high-street banks in order for the companies' certificates to be physically uploaded. Although this makes the system more cumbersome to use, it does give users reason to believe that the certificates on their device are only from reputable companies, addressing the redirection problem.

5.2 HTML Rewriting and Active Injection Attacks

Since we assume that the PC may be entirely compromised, crimeware has the capability to rewrite the HTML in the PC's browser—e.g. swapping link targets around, adding new links, modifying text. We address this issue with reference to our Security Policy Model (Section 2.2). From points 4 and 5 of the Security Policy Model we know that even if the user is fooled into initiating a security-sensitive operation due an HTML-rewriting attack, all that will happen is that a fully-specified confirmation dialogue appears on their trusted personal device. If the user does not confirm the action via the trusted personal device the web-application will not carry it out. Similarly, since points 1-3 of the Security Policy Model require the web application to encrypt all security-sensitive content, an HTML rewriting attack cannot cause this information to be revealed.

The problem of active-injection (see Section 2.1) is dealt with in the same way. If crimeware on the untrusted PC malliciously attempts to initiate a security-sensitive operation (say, by spoofing a click on a hyperlink) then our Security Policy Model dictates (*i*) that no security-sensitive operations will be performed without first requesting confirmation via the trusted personal device; and (*ii*) that, since security sensitive information is always encrypted, it will not be revealed.

Another form of HTML rewriting attack relates to form submission. In this case the untrusted PC may maliciously put an encrypted user-input message received from the personal device into the wrong form field before completing a form submission (see Section 3.5.1). The aim of this attack may be to fool the web application into binding the wrong piece of user-input to the wrong form field. Recall (from Section 3.5.1) that we deal with this attack by ensuring that encrypted user-input messages generated by the trusted personal device contain $(\langle \textit{fieldname} \rangle, \langle \textit{user-input} \rangle)$ pairs, which are parsed by the web application. Since crimeware on the untrusted PC cannot change the content of the encrypted messages it cannot cause the wrong piece of user-input to be associated with the wrong form field. Also, in accordance with point 4 of our Security Policy Model, we ensure that crimeware cannot replay form submissions (see Section 3.5.2).

It is worth observing that attacks against the RDC Agent directly are really just special cases of HTML-rewriting/active injection attacks.

[7] Although password-hashing can be implemented directly on the untrusted PC [24] this does not protect against OS-level keylogging attacks. Thus we would implement password-hashing on the trusted device.

5.3 Message Re-Ordering Attacks

A major difference between our architecture and conventional secure transport protocols (such as SSH [31]) is that we do not embed *sequence numbers* in encrypted messages. A man-in-the-middle (including, of course, crimeware on the untrusted PC) is thus able to re-order the messages in transit between the web-application and the trusted personal device.

Our omission of a sequence number is quite deliberate; it would provide no additional security in the context of our architecture. The reason for this is because crimeware on the untrusted PC is already capable of mounting active-injection attacks. Why bother to preserve the order in which packets were sent by the web-application when the order in which they were *requested* can be spoofed so easily? Instead, we observe that message re-ordering attacks are just a subset of HTML rewriting and active-injection attacks, and address them in the same manner: not by preventing them from happening, but by designing web-applications in such a way that it does not matter if they do happen—i.e. with reference to our Security Policy Model.

As a brief aside, note that one may propose an alternative split-trust web-browsing framework in which all clicks on hyperlinks are initiated (or somehow confirmed) on the personal device. In this context, SSH-style sequence numbering would provide some value, since the order in which the web-application sends its messages is worth preserving. However, the downside of this scheme is that the requirement to initiate/confirm *all* clicks via the personal device would make the system cumbersome to use. Thus, we argue that our Security Policy Model finds a sweet-spot on the security-usability spectrum for split-trust applications.

5.4 Social Engineering Attacks

Split-trust browsing requires users to understand a simple principle: trust your personal device, not the PC. However, attackers may conspire to make users doubt this principle causing them (say) to unwittingly confirm a security-sensitive operation via their trusted personal device.

For example, the untrusted PC may perform an HTML-rewriting attack, malliciously adding the text "you will now see a confirmation dialogue appearing on your personal device; please click confirm". At the same time, it may use an active-injection attack to initiate a security-sensitive operation. The question is, when the confirmation dialogue appears on their personal device, will users remember the "trust your personal device, not the PC" principle, or will they be fooled into clicking on confirm?

We believe that this type of attack is dangerous—the success of phishers suggests that some users will always be duped by this kind of ploy. However, although split-trust browsing is not fool proof against attacks of this nature, it still raises the bar. Without split-trust browsing, an active-injection attack perpetrated by crimeware running on the PC would simply result in a security-sensitive operation being performed or secret information being revealed—the user would not have any chance to prevent it. With split-trust browsing crimeware has to simultaneously initiate the security-sensitive operation *and* successfully fool the user into OK-ing the fully-specified confirmation dialogue on their phone.

Extensive user testing is required to determine how users of split-trust web applications may respond to this type of attack.

6 Discussion and Related Work

The idea of simultaneously using multiple devices to access applications and data has been explored extensively by the research community [19, 23]. Our work adopts these ideas, using them to protect against PC-based crimeware attacks. We are also influenced by the *Situated Mobility* [22, 30] and *Parasitic Comput-*

ing [20] models of ubiquitous computing, in which small mobile devices (e.g. cell phones) co-opt computing resources already present in the environment (e.g. public screens) to facilitate interaction with their users.

The idea of split-trust is not new. Balfanz and Felton demonstrated the idea of splitting an application between a trusted PDA and untrusted PC in the context of an email signing application [7]. Other researchers have since applied this technique to thin-client-based mobile computing [21]. The main technical contribution of our work is to explore the security and systems issues surrounding a general framework for split-trust web applications. However, we believe that we have made other contributions to split-trust engineering more generally: we are the first to consider the extent to which a split-trust application can protect against active-injection attacks, to propose a Security Policy Model for split-trust systems and to consider potential social-engineering attacks against split-trust technology (see Section 5.4).

Ross *et al.* developed a web-proxy which detects security-sensitive words and phrases in HTML content, replacing them with codewords. Users can simultaneously connect their PDA to the proxy in order to download a mapping from codewords back to their original text [25]. Ross' work does not allow HTML to be split generally and, most critically, does not allow data-entry to be performed via the PDA; as a result his system does not protect against keylogging and active injection attacks. We believe our architecture for splitting HTML generally, our ability to migrate user-input to the trusted personal device to avoid PC-based keylogging attacks, and our Security Policy Model for generalised split-trust web-applications is a significant advance on Ross' work.

Ross' web-proxy [25], and other previous work on split-trust architectures [21, 28] require the personal device to open a dedicated Internet connection to a trusted server. In contrast, one of the interesting aspects of our split-trust framework for web applications is that we are able to embed encrypted messages in the untrusted PC's HTML, relying on the RDC Agent to demultiplex these two logical channels. Although it does not affect the security properties of the system, we believe that this approach leads to significant usability benefits (for the four reasons listed in Section 1.2).

6.1 What Makes a Personal Device Trusted?

Ideally, one could imagine designing and manufacturing trusted personal devices specifically for split-trust browsing. Such devices could be technically very simple supporting only basic I/O capability, a data-link technology that enables direct connection to a PC (e.g. USB or Bluetooth), cryptographic functionality and a stripped down cHTML browser. A security-focused design from the outset, combined with its technical simplicity could make such a product a significantly more trusted platform than a modern general purpose PC.

From a more pragmatic perspective, some security researchers claim that some existing cell phones and PDAs already provide a more trusted computing platform than general purpose PCs [7, 21]. In particular: (*i*) users only rarely install privileged applications on their phones[8] reducing the risk of trojan-based crimeware; and (*ii*) whereas it is often easy for attackers to gain physical access to PCs[9] in order to install crimeware, it is much harder to gain physical access to a users' cellphone.

Thus, whilst the best trusted personal devices would be designed specifically for that purpose from the outset, we believe that, in the short term, users could still benefit from split-trust browsing with their existing PDAs or cell phones. (We note that the architecture presented in this paper is applicable regardless of the implementation details of trusted personal devices.)

A number of manufacturers are starting to incorporate hardware into cell phones specifically to provide strict process isolation and to manage encryption keys/private data [4, 11]. We see this as a promising sign, suggesting that security is increasingly being seen as an important aspect of mobile computing devices. Such

[8] Many phone applications that users install are sandboxed Java MIDP applets that are not capable of general keylogging or screengrabbing.

[9] Consider a public terminal in an Internet cafe, or even a PC left unattended and unlocked in an office environment.

technology has the potential to isolate trusted mobile applications (such as application-support required for split-trust browsing) from the effects of mobile phone viruses [8] and malicious code.

6.2 Generalising our Architecture

The architecture presented in Section 3 is just one of a number of possible implementation alternatives, each with their own advantages and shortcomings. For example, we may have chosen to implement the RDC Agent as an HTTP proxy that runs as a native process on the PC. This has the benefit of enabling one RDC Agent to work with multiple browsers; however, it makes it more difficult for the RDC Agent to respond to user-interface events occurring within the browser[10]. Similarly, we may have chosen a different embedding strategy for messages destined for the trusted personal device, or a different mechanism for co-ordinating key exchange. The purpose of Section 3 is thus not to present the definitive architecture for split-trust browsing, but instead to demonstrate that such an architecture can be built on top of existing infrastructure whilst achieving acceptable performance.

There are a number of ways that the architecture presented in this paper could be generalised. For example, in its current form, the trusted personal device only stores one session key at a time; thus, when a new split-trust session starts, the previous one is automatically closed. To avoid this we could borrow from SSL client design, enabling the trusted personal device to maintain a table of active session keys indexed by the domain of the current URL.

There are also a number of places where the mechanism for splitting content between PC and personal device could be generalised. For example, our current implementation does not allow images to appear on the personal device. This functionality could be added (say) by allowing image data to be embedded directly in the cHTML forwarded to the personal device. Similarly, one may wish to allow hyperlinks to appear on the trusted personal device (a feature which our current architecture does not allow).

Of course, it is unclear whether these generalisations would have a positive or a negative effect on the overall usability of the system. Further research is required to answer such questions.

6.3 Server Side Programming

To enable rapid authoring of split-trust web applications there is clearly a need for a server-side API which abstracts the details of key-exchange, message encryption and the embedding of messages in HTML from web programmers. This API could be provided as a set of functions available in a variety of languages commonly used to write web applications (e.g. Java, PHP etc.). Alternatively, one could envisage a scheme in which web-application designers insert HTML tags to mark up the content that is to appear on the trusted personal device (e.g. `<personaldev> ...</personaldev>`). A server-side HTML-rewriting proxy could then be employed to detect these tags, automatically encrypting their bodies with the appropriate session key. In many ways this approach is analogous to the previous work on abstracting security policies from complex web applications [27].

6.4 Usability Issues

Although this is primarily a systems-security paper, there were some usability issues that came to light during our implementation work which we choose to document here:

1. On the PC screen there is a clear need to visually differentiate between links that cause new content to appear on the PC and links that cause new content to appear on the personal device. To address

[10] For example, the RDC Agent presented in Section 3.2 works well with Firefox's tabbed browsing—when the user clicks on a different tab, the RDC Agent traps this event and forwards the new page's `rdc-onLoad-msg` to the user's personal device (or clears the personal device's screen if there is no `rdc-onLoad-msg`).

this issue we used a stylesheet that defined a class of "personal device link", rendering them with a highlighted background. A web application uses the `class` attribute to mark these links (see Figure 4).

2. The factor that we found made the most significant difference to usability is at first a seemingly trivial concern: the ability to stick the personal device on the side of the PC monitor. This enables both hands to be free for mouse/keyboard input; furthermore, the proximity between the PC display and the phone display enables the user to keep them both in their peripheral vision simultaneously. As a result, the user experiences virtually no overhead in managing the two displays: instead, they are able to treat the two logical displays as one.

3. A number of users were concerned about what would happen if they didn't have a trusted personal device. Of course, the answer to this question depends entirely on the service-provider's wishes. For example, a web-application may deny access unless a trusted personal device is available; alternatively, a PC-only version of the site could be provided that gives access to the less security-sensitive parts of the UI.

7 Conclusions

Crimeware is becoming a serious problem, threatening to take over from phishing as the dominant form of cyber-crime in the not too distant future [1]. The web's security model (HTTPS/SSL) protects data as it is transmitted between client and server, but does not prevent crimeware attacks in which the end-points themselves are compromised.

In this paper we have proposed a technique in which users can combine their PC with a trusted personal device to defeat crimeware (Section 1). Information entered via the personal device's keypad cannot be read by the PC, thwarting PC-based keyloggers. Similarly, information displayed on the personal device's screen is also hidden from the PC, preserving the confidentiality and integrity of security-critical data even in the presence of screengrabbing attacks and compromised PC browsers. We aim to provide users with the "best of both worlds": they can simultaneously enjoy the rich interaction capabilities of their PC, with its large display and full-sized keyboard *and* the greater degree of trust associated with viewing/entering security-sensitive data via their personal device.

We presented an architecture for split-trust browsing which is based entirely on existing, deployed technology (Section 3) and demonstrated that acceptable performance is achievable using today's personal devices (Section 3.6). In our subsequent discussion (Section 6) we considered how this architecture may be generalised, what may constitute a *trusted* personal device and highlighted the necessity for a split-trust server-side programming model—something we intend to concentrate on in future work.

Previous work on split-trust systems [25, 21, 7] has not considered the general design principles that enable applications to minimise trust in the client PC. We believe that our Security Policy Model is an important contribution in this respect. Whereas Oprea *et. al.* admit that they are forced to "*trust [the client PC] to a certain extent*" [21], our Security Policy Model demonstrates that it is possible to design split-trust applications that do not trust the client PC at all.

In future work we are particularly keen to explore the hardware/software design of a secure personal device targeted specifically at split-trust browsing (see Section 6.1). The insights gained from this process will reveal design lessons for future mobile computing platforms. We also intend to carry out more extensive usability testing, particularly surrounding some of the social-engineering attacks that we believe split-trust systems in general may be susceptible to (see Section 5.4).

As we have discussed in Section 5, split-trust web browsing is not a panacea. However, we do believe that it has the potential to provide consumers with a significantly greater degree of security in the face of

ever-increasing crimeware and phishing attacks. Of course, our system delivers value proportional to the security of the trusted personal devices employed. It is our hope, therefore, that by presenting application scenarios for secure mobile computing, split-trust research motivates vendors to incorporate security-enhancing technologies (e.g. ARM's TrustZone [4] and Intel's Mobile iA [11]) into personal devices.

References

[1] Anti-Phishing Working Group expands online identity theft charter. August 3rd 2005 edition of Business Wire. Available from `http://www.businesswire.com/`.

[2] The Bluetooth Specification version 1.1. `http://www.bluetooth.com/`.

[3] XMLHTTP. `http://en.wikipedia.org/wiki/XMLHttpRequest/`.

[4] ALVES, T., AND FELTON, D. TrustZone: Integrated hardware and software security, July 2004. ARM White Paper.

[5] ANDERSON, R., STAJANO, F., AND LEE, J.-H. Security policies. In *Advances in Computers vol 55* (2001), Academic Press.

[6] ANTI-PHISHING WORKING GROUP (APWG). Phishing activity trends report, June 2005. `http://antiphishing.org/`.

[7] BALFANZ, D., AND FELTON, E. Hand-held computers can be better smart cards. In *Proceedings of USENIX Security* (1999).

[8] BBC NEWS. First mobile phone virus created. `http://news.bbc.co.uk/1/hi/technology/3809855.stm`.

[9] BOSWELL, D., KING, B., OESCHGER, I., COLLINS, P., AND MURPHY, E. *Creating Applications with Mozilla*. O'Reilly, 2002.

[10] CHOWN, P. Advanced Encryption Standard (AES) ciphersuites for Transport Layer Security (TLS). RFC 3268.

[11] COLE, B. Intel hardwires security in new mobile iA PXA27x CPU family. `http://iapplianceweb.com/story/OEG20040412N0006BC.htm`.

[12] DIERKS, T. The tls protocol.

[13] FLANAGAN, D. *JavaScript: The Definitive Guide*. O'Reilly, 2002.

[14] HARKINS, D., AND CARREL, D. The Internet Key Exchange. RFC 2409.

[15] JOSEFSSON, S. The Base16, Base32, and Base64 data encodings. RFC 3548.

[16] KAMADA, T. Compact HTML for small information appliances, 1998. W3C Note. Available from `http://www.w3.org/TR/1998/NOTE-compactHTML-19980209/`.

[17] LECLAIRE, J. Pharming and SPIM plaguing Internet. 4th June 2005. TechNewsWorld. Available from `http://www.technewsworld.com/story/news/42054.html`.

[18] LEYDEN, J. UK police issue "vicious" trojan alert. 13th August 2004. The Register. Available from `http://www.theregister.co.uk/2004/08/13/trojan_phish/`.

[19] MYERS, B. A. Using handhelds and PCs together. *Commun. ACM 44*, 11 (2001), 34–41.

[20] NARAYANASWAMI, C., RAGHUNATH, M. T., KAMIJOH, N., AND INOUE, T. What would you do with 100 MIPS on your wrist? Tech. Rep. RC 22057 (98634), IBM Research, January 2001.

[21] OPREA, A., BALFANZ, D., DURFEE, G., AND SMETTERS, D. Securing a remote terminal application with a mobile trusted device. In *Proceedings of ACSA 2004*. Available from `http://www.acsa-admin.org/`.

[22] PERING, T., AND KOZUCH, M. Situated mobility: Using situated displays to support mobile activities. In *Public and Situated Displays: Social and Interactional Aspects of Shared Display Technologies* (2003), Kluwer.

[23] RAGHUNATH, M., NARAYANASWAMI, C., AND PINHANEZ, C. Fostering a symbiotic handheld environment. *Computer 36*, 9 (2003), 56–65.

[24] ROSS, B., JACKSON, C., MIYAKE, N., BONEH, D., AND MITCHELL, J. C. Stronger password authentication using browser extensions. In *Proceedings of the USENIX Security Symposium* (2005), USENIX association.

[25] ROSS, S. J., HILL, J. L., CHEN, M. Y., JOSEPH, A. D., CULLER, D. E., AND BREWER, E. A. A composable framework for secure multi-modal access to Internet services from Post-PC devices. *Mob. Netw. Appl. 7*, 5 (2002), 389–406.

[26] SCHNEIER, B. *Applied cryptography: protocols, algorithms, and sourcecode in C*. John Wiley & Sons, New York, 1994.

[27] SCOTT, D., AND SHARP, R. Abstracting application-level web security. In *Proceedings of ACM WWW* (2002), ACM Press.

[28] SHARP, R., SCOTT, J., AND BERESFORD, A. Secure mobile computing via public terminals. To appear in proceedings of PERVASIVE 2006.

[29] SOPHOS PRESS RELEASE. UK online bank accounts put at risk by new trojan. Available from `http://www.sophos.com/virusinfo/articles/ukbanktrojan.html`.

[30] WANT, R., PERING, T., DANNEELS, G., KUMAR, M., SUNDAR, M., AND LIGHT, J. The personal server: Changing the way we think about ubiquitous computing. In *UbiComp '02: Proceedings of the 4th international conference on Ubiquitous Computing* (London, UK, 2002), Springer-Verlag, pp. 194–209.

[31] YLONEN, T. SSH transport layer protocol. RFC 4253.