RC24532 (W0804-035) April 9, 2008
Computer Science

IBM Research Report

Trustworthy and Personalized Computing on Public Kiosks

Scott Garriss[1], Ramón Cáceres[2], Stefan Berger[3], Reiner Sailer[3], Leendert van Doorn[4], Xiaolan Zhang[3]

[1]Carnegie Mellon University
Pittsburgh, PA

2AT&T Labs
Florham Park, NJ

[3]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

[4]AMD
Austin, TX

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Trustworthy and Personalized Computing on Public Kiosks

Scott Garriss
Carnegie Mellon University
Pittsburgh, PA, USA

Ramón Cáceres
AT&T Labs
Florham Park, NJ, USA

Stefan Berger
IBM T. J. Watson Research Center
Hawthorne, NY, USA

Reiner Sailer
IBM T. J. Watson Research Center
Hawthorne, NY, USA

Leendert van Doorn
AMD
Austin, TX, USA

Xiaolan Zhang
IBM T. J. Watson Research Center
Hawthorne, NY, USA

ABSTRACT

Many people desire ubiquitous access to their personal computing environments. We present a system in which a user leverages a personal mobile device to establish trust in a public computing device, or *kiosk*, prior to resuming her environment on the kiosk. We have designed a protocol by which the mobile device determines the identity and integrity of all software loaded on the kiosk, in order to inform the user whether the kiosk is trustworthy. Our system exploits emerging hardware security technologies, namely the Trusted Platform Module and new support in x86 processors for establishing a dynamic root of trust. We have demonstrated the viability of our approach by implementing and evaluating our system on commodity hardware. Through a brief survey, we found that respondents are generally willing to endure a delay in exchange for an increased assurance of data privacy, and that the delay incurred by our unoptimized prototype is close to the range tolerable to the respondents. We have focused on allowing the user to personalize a kiosk by running her own virtual machine there. However, our work is generally applicable to establishing trust on public computing devices before revealing any sensitive information to those devices.

Categories and Subject Descriptors

C.5.3 [**Microcomputers**]: Portable devices; K.4.1 [**Public Policy Issues**]: Privacy; K.6.5 [**Security and Protection**]: Invasive software and Unauthorized Access; K.8.m [**Personal Computing**]: Miscellaneous

General Terms

Security, Human Factors, Verification

Keywords

Kiosk computing, trusted platform module, mobility, personalized computing, integrity verification, virtualization

1. INTRODUCTION

Public computing *kiosks*, such as a rental computer at an Internet café, are widely used because they provide users a convenient and powerful computing platform without requiring them to carry cumbersome and costly portable computers such as laptops. Unfortunately, being general-purpose computers, kiosks are susceptible to a wide variety of attacks that can compromise the privacy of the user's data and the integrity of her computations. Consequently, the user would like assurance that a kiosk has not been compromised before entrusting to it any personal data or computations. The owner of a kiosk similarly wants to ensure that it is not used to perform malicious acts for which he may be liable. Current kiosks do not provide such assurances.

Securing a public kiosk is a daunting task. An attacker may, for example, install a malicious program such as a keystroke logger, insert a rogue virtual machine monitor (VMM) that can observe and modify an otherwise legitimate software environment, compromise modifiable software or firmware such as the BIOS, or add malicious hardware such as a USB sniffer. Each of these attacks poses difficult challenges. In this work we focus on detecting any malicious software loaded by the kiosk, including a rogue VMM and a compromised BIOS. Software attacks are far more common than hardware attacks and an important threat in their own right. We believe that our work meaningfully increases the level of trust that users can place in kiosks.

This paper presents a system, depicted in Figure 1, in which a user leverages a personal mobile device, such as a smartphone, to gain a degree of trust in a kiosk prior to using the kiosk. In the context of computer systems, trust is the expectation that a system will faithfully perform its intended purpose [10]. We refer to a kiosk as trustworthy if we have verified the identity and integrity of all software loaded on it. Our system aims to increase trustworthiness without compromising functionality—the user should be able to use the full computing and I/O capabilities of the kiosk. We assume that the mobile device is a priori trustworthy. Mobile device security is itself an important issue that we do not address here except to argue, as others have done [4, 21, 26], that a user can place a greater degree of trust in a personal mobile device than in a general-purpose computer or public kiosk.

We have designed a protocol by which the mobile device verifies that the kiosk has loaded only trustworthy software. The user's involvement in the protocol is limited to using

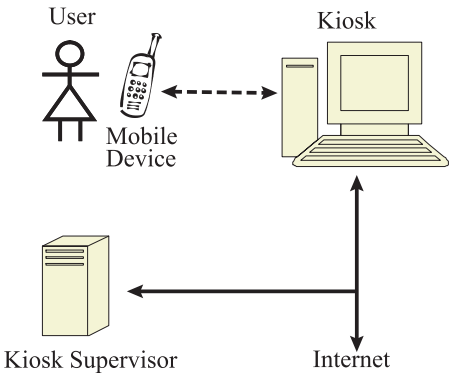

Figure 1: Kiosk computing scenario

the device to take a photo of a barcode, choose a desired software configuration, and view the result of the protocol. If the protocol succeeds, the user can proceed to use the kiosk. If the protocol fails, the user can walk away. Our design also allows a supervisor machine, acting on behalf of the kiosk owner, to use a subset of the protocol to verify that the kiosk has loaded only approved software. If unapproved software is found, the owner can disable the kiosk, e.g., by removing it from the network.

Our system employs a number of emerging security technologies. We utilize new x86 processor support for establishing a dynamic root of trust on commodity platforms that incorporate AMD's Secure Virtual Machine Technology [8] or Intel's Trusted Execution Technology [11]. We also use the Trusted Platform Module (TPM) [10] together with the Integrity Measurement Architecture (IMA) [23] to provide both user and owner with proof that only trustworthy software has been loaded.

We focus on using virtual machines to enable the user to resume a complete personal computing environment that includes her choice of operating system, applications, settings, and data. Internet Suspend/Resume (ISR) [14] and SoulPad [7] are two solutions based on running users' virtual machines on kiosks. Our work resolves important security issues left open by both approaches. In particular, we eliminate blind trust in any software component on the kiosk, including the BIOS, and we prevent the attack where a rogue virtual machine monitor runs below an unsuspecting user environment [13].

Our main contributions are the following:

- The design of a protocol by which a user leverages a personal mobile device to establish trust on a public computing device prior to revealing any personal information to the public device (Section 4).

- The implementation, using commodity components, of a trustworthy kiosk computing system that embodies this design (Section 5).

In a short position paper [9], we previously presented a preliminary design and implementation of our system, and identified a number of open issues. This paper improves upon our earlier work by presenting a more complete design, implementation, and evaluation of our system. In particular, this paper additionally presents:

- A practical procedure for distributing a database of fingerprints of trusted software components (Section 4.2).

- A solution to the kiosk-in-the-middle attack that uses a visual channel to identify the particular kiosk in front of the user (Section 4.3.1).

- How to set up a secure channel between the phone and kiosk for the transmission of private data (Section 4.3.4).

- How to securely establish a personalized computing environment on the kiosk by resuming a virtual machine there (Sections 4.3.6 and 5.3).

- A procedure for purging any personal data that may reside on a kiosk when a user finishes using it (Section 4.3.7).

- An evaluation of user expectations in the kiosk scenario together with relevant performance characteristics of our prototype system (Section 6).

2. USER EXPERIENCE

To provide context for the rest of this paper, we begin with an overview of how a user interacts with our trustworthy kiosk computing system. We believe this interaction is simple enough that it does not place an undue burden on the user while providing important security functionality.

Figure 2 shows a timeline of the steps a user follows after stepping up to a kiosk that she intends to use. First, she makes the identity of the kiosk known to her mobile phone. We propose for kiosk owners to display a numerical identifier on the outside of the kiosk in barcode form. The user captures the contents of the barcode using the digital camera available on modern phones [17].

Second, the phone presents the user with a list of software configurations available on the kiosk. Figure 3(a) is an example of a mobile phone screen that presents these choices to the user. In this example, there are two choices, a personalized computing environment to be provided by the user, or a standard set of applications provided by the kiosk owner. The user selects the configuration she wants to use, and the phone forwards the choice to the kiosk.

Third, the user simply waits while the phone and kiosk carry out the rest of our trust establishment protocol, which we will describe in detail in Section 4. At the completion of this protocol, the phone announces to the user that the kiosk is either untrustworthy or trustworthy. Figures 3(b) and 3(c) show examples of these two cases. If the kiosk is declared untrustworthy, the user can walk away before divulging any personal information to the kiosk. If the kiosk is declared trustworthy, the user can proceed to use the kiosk.

Fourth, Figure 2 depicts what we consider to be a particularly compelling example of kiosk computing, namely when the user runs a personal virtual machine on the kiosk. Virtual machines enable users to run complete and highly customized computing environments on a wide range of hardware machines, including public kiosks. However, a virtual machine can contain a great deal of personal information, and therefore the user should only run virtual machines on trustworthy kiosks. Figure 3(d) shows a mobile phone screen that gives the user a choice of virtual machines to run on the kiosk. This step is unnecessary if the user earlier chose to use standard software provided by the kiosk instead of a personalized environment.

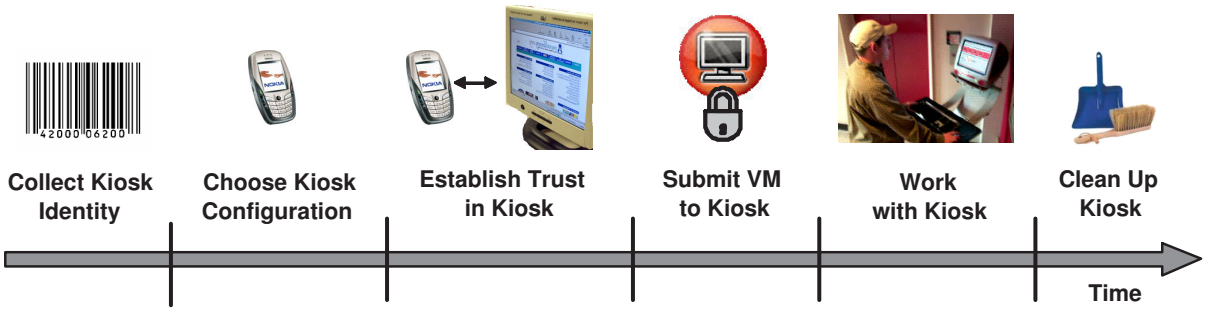

Figure 2: Timeline for use of a trustworthy and personalized kiosk

(a) (b) (c) (d)

Figure 3: Mobile device screens seen by a user when interacting with a kiosk

Fifth, the user proceeds to work on the kiosk. Finally, the user logs out of the kiosk, which triggers cleanup operations to make the kiosk ready for the next user.

3. BACKGROUND

This section provides technological background that is useful in understanding the rest of the paper.

Trusted Platform Module (TPM).

The TPM [10] is a hardware component that is increasingly available in personal computers and servers at a fraction of the cost of other secure hardware, such as a coprocessor. It provides a variety of security functions, including cryptographic primitives, such as signatures, and secure storage for small amounts of data, such as cryptographic keys. The TPM is resistant to software attacks because it is implemented in hardware and presents a carefully designed interface.

Especially notable is the TPM's ability to store cryptographic hashes, or *measurements*, of software components in a set of Platform Configuration Registers (PCRs). PCRs are initialized at boot and may not be otherwise reset, with one important exception described below. They may only be modified via the *extend* operation, which takes an input value, appends it to the existing value of the PCR, and stores the SHA1 hash of the result in the PCR. A single PCR can thus store an aggregate representation of an arbitrary sequence of software components. The cryptographic properties of this operation state that it is infeasible to reach the same PCR state through different sequences of inputs.

This technique allows the TPM to guarantee the load-time integrity of running software. This is sufficient to detect the execution of malicious software (e.g., a keystroke logger), but the TPM cannot detect compromises, such as buffer overflow attacks, that occur after software is loaded. Providing strong run-time guarantees is an area of active research (cf. [25]).

Each TPM has a variety of associated keys; we limit our discussion to the asymmetric Attestation Identity Key (AIK). The TPM generates this keypair, and stores the private key in internal protected storage. The private AIK is used to sign quotes that attest to the current state of the TPM's PCRs. The public AIK is included in an AIK certificate that provides a binding between a public key and a certified-legitimate TPM. The reliance on a certification authority is a commonly perceived weakness of current TPM implementations.

Integrity Measurement Architecture (IMA).

The TPM may be used to achieve *trusted boot*, where measurements stored in PCRs are used to verify that the software stack through the OS kernel meets expectations. IMA [23] extends trusted boot by having the OS additionally measure applications and configuration files. IMA maintains in software a *measurement list* containing a text description and the corresponding hash value of each software component that has been measured into the TPM.

IMA further provides an *attestation protocol* that allows a remote IMA *verifier* to challenge the integrity of an IMA platform. The verifier first sends an attestation request to the IMA *attestation server*, which then replies with the current measurement list, along with a *quote* containing an aggregate of the current PCR values, signed by the TPM. The verifier then uses the measurement list to replay the sequence of PCR extend operations and verify that the resulting aggregate PCR value agrees with the signed quote. Finally, the verifier compares the measurement list to a

measurement database of known software, thus verifying the identity and integrity of software on the challenged system.

Dynamic Root of Trust for Measurement (DRTM).
As mentioned, general PCRs are initialized at boot time and cannot be reset. Trusted boot uses these PCRs to establish a *static* root of trust, which must include all software loaded since boot, starting with the BIOS. Recent extensions to the x86 architecture support the establishment of a *dynamic* root of trust by allowing a special PCR (PCR 17) to be reset at any time by a special CPU instruction, `skinit` in AMD processors and `senter` in Intel processors. This instruction takes as input a 64KB section of code known as the *secure loader*, and places the processor in a special state that guarantees atomicity, e.g., by disabling interrupts. The instruction then resets PCR 17, measures the secure loader, extends PCR 17 with this measurement, and transfers control of the processor to the secure loader.

These three technologies offer distinct benefits in our kiosk computing scenario. The TPM provides a hardware root of trust and is thus resistant to software attacks. IMA allows the kiosk to use the TPM to prove to the user's phone that the kiosk has loaded a particular software stack. A DRTM allows us to remove the BIOS and boot loader from the set of software that must be measured.

4. SYSTEM DESIGN

This section presents the design of our trustworthy kiosk system, in particular the trust establishment protocol that is central to our approach. As shown in Figure 1, our system consists of a user carrying a mobile device, a kiosk, and a kiosk supervisor. The mobile device is pre-equipped with an application that aids the user in ascertaining the trustworthiness of the kiosk. This application incorporates an IMA verifier. The kiosk is a PC-class platform equipped with a DRTM-enabled processor and a TPM. The kiosk will additionally run an IMA attestation server and a front-end for talking to the mobile device. The kiosk supervisor may be any platform capable of running an IMA verifier.

4.1 Threat Model

Our system aims to protect the confidentiality of the user's data in an environment where an attacker is located in close physical proximity to the kiosk, and may eavesdrop on or inject messages into any wireless communication between the mobile device and the kiosk. The attacker may have unfettered access to the kiosk before the user arrives and after the user leaves (including knowledge of the root password), and has computational and communication resources roughly equivalent to that of a desktop PC (i.e., insufficient to break any underlying cryptographic primitives).

The user must trust the kiosk owner to not maliciously modify the kiosk's hardware. Furthermore, she must trust the owner to periodically inspect kiosks to ensure that they remain unmodified. The user trusts the owner's intention to install only benign software on the kiosk, but the TPM allows her to verify that the owner has done so correctly.

4.2 Measurement Database

The IMA verifier on the mobile device requires access to a database containing the expected measurements for all software loaded on the kiosk. In our system, the kiosk provides the mobile device with this database. This database is signed by a trusted third party, such as the kiosk's vendor, who certifies the trustworthiness of the software identified in the database. This process assumes that the mobile device knows the public key of the trusted third party, or can readily obtain it via a public-key infrastructure.

Maintaining a set of IMA databases for general computing is onerous because of the variety of software components and versions that must be tracked. However, several aspects of our system combine to mitigate this problem. First, kiosks often have restricted functionality (e.g., a kiosk that loads only a VMM), which reduces the number of software components that must be tracked, especially applications. Second, establishing a DRTM after the BIOS loads eliminates the BIOS from the Trusted Computing Base (TCB), which in addition to gaining the security benefits of a smaller TCB, implies that the IMA verifier need not track BIOS versions across different kiosks. Despite being small and relatively stable, the BIOS poses a unique challenge in that it varies widely between machines. Third, since the kiosk owner is responsible for aggregating the requisite database information, he need only collect information pertaining to his kiosks, which are likely to have similar configurations.

Whenever the kiosk owner updates the kiosk software, the owner can identify the installed software to the third party, who can respond with a signed database containing only the software present on the kiosk. Since the third party, through prior communication with software vendors, is likely to already know the measurements for the software installed on the kiosk, a signed kiosk-specific database can be created using an automated process.

4.3 Trust Establishment Protocol

Figure 4 presents our protocol for establishing trust in a kiosk. The protocol roughly consists of six phases. In the first phase (Steps 0–3), the phone obtains the public key that can be used to verify attestations from the kiosk's TPM. In the second phase (Steps 4–5), the phone demonstrates to the kiosk that it is authorized to use the kiosk. In the third phase (Steps 6–7), the phone and kiosk decide which software configuration the kiosk should boot. In the fourth phase (Step 8), the kiosk reboots, establishes a dynamic root of trust, and loads the desired software. In the fifth phase (Steps 9–14), the phone utilizes the IMA protocol to verify the integrity of the kiosk's software using the public key obtained in the first phase. If this succeeds, the phone will deem the kiosk trustworthy, and continue to the final phase, in which the user interacts with the kiosk (Steps 15–16).

4.3.1 Obtain the public key of the kiosk

Establishing secure communication between two wireless devices is challenging because the user has little evidence indicating *which* device is on the other end of the connection. The solution we adopt is that of McCune et al. [17] in which the visual channel provided by the phone's camera is used to securely obtain the address and public key of the kiosk. Section 7 discusses the security ramifications of this approach. First, the user photographs a barcode affixed to the kiosk (Step 0). This barcode encodes both the Bluetooth address of the kiosk and the hash of the Attestation Identity Key (AIK) certificate of the kiosk. The AIK certificate includes the public AIK, whose private counterpart the kiosk's TPM will use to sign integrity measurements later in the protocol.

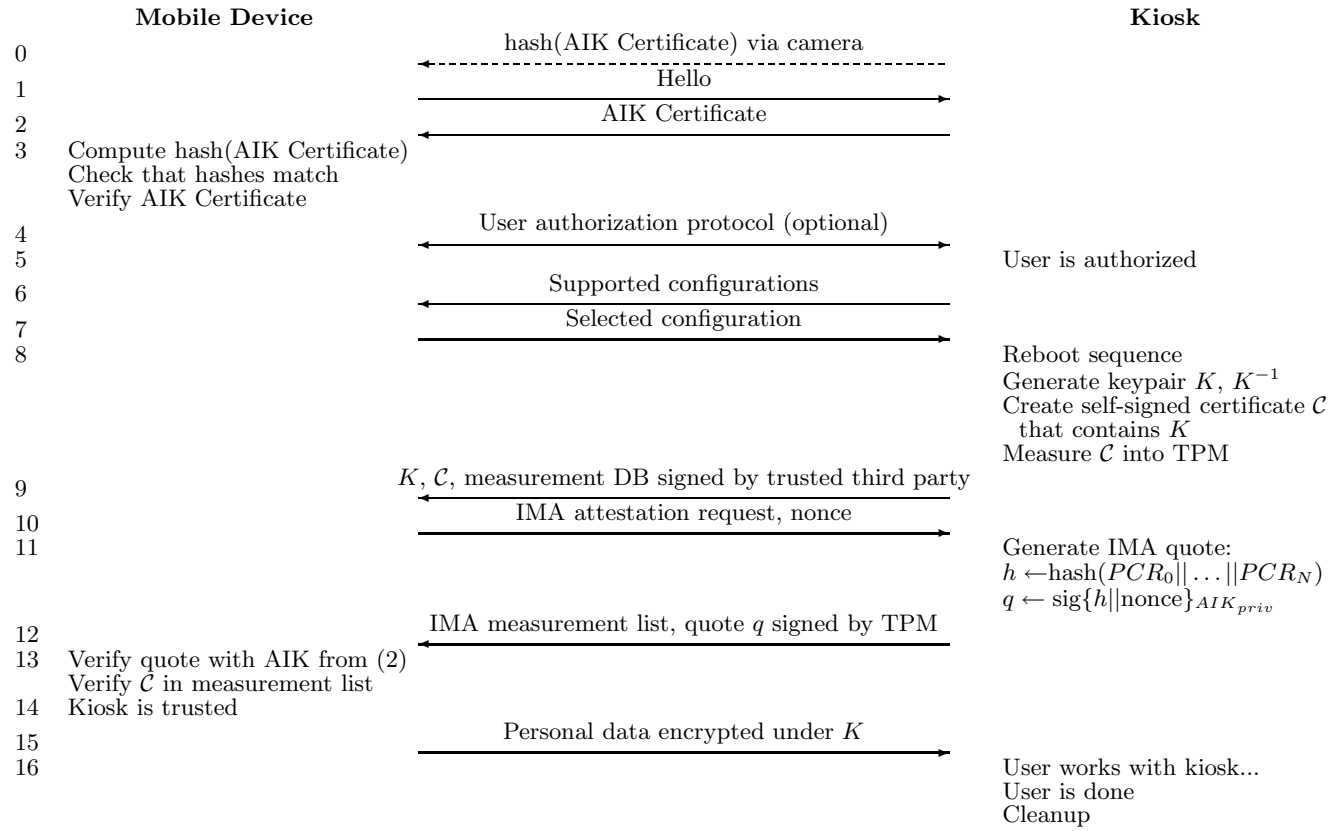

Figure 4: Trust establishment protocol between mobile device and kiosk

After capturing the barcode, the phone will initiate the protocol with the kiosk (Step 1) using the Bluetooth address obtained from the barcode. The kiosk then transmits its AIK certificate over the wireless channel to the phone, who verifies the signature on the certificate and that the hash of the certificate matches the barcode. With current TPM implementations, verifying the AIK certificate's signature requires the phone to know the public key of the certification authority (known as the *Privacy CA*) that issued the certificate. At the conclusion of this phase, the phone knows the public AIK that the kiosk's TPM will use to sign integrity measurements and that the TPM is legitimate.

4.3.2 Demonstrate authority to use the kiosk

In scenarios where kiosk use is restricted to certain individuals (e.g., paying customers), Step 4 allows the user to demonstrate authority to use the kiosk. Since the kiosk is not yet trusted, this scheme should not reveal private information about the user. Anonymous proof of payment [6] is one possibility. Free public kiosks may omit Steps 4 and 5. At the conclusion of this phase, the kiosk has determined that the user is authorized.

4.3.3 Select desired configuration

In Step 6, the kiosk provides the phone with a list of configurations that it is prepared to boot. The phone then prompts the user to select the desired configuration and forwards the choice to the kiosk.

4.3.4 Reboot

Figure 5 shows the boot sequence for our kiosk, which differs from a standard boot sequence by the addition of Steps 2 and 3. Step 2 executes the skinit or senter instruction to establish a dynamic root of trust and extend a designated PCR with the digest of the secure loader. Control of the processor is then transferred to the secure loader (Step 3), which extends a PCR with the digest of the relevant hypervisor and OS kernel files. The secure loader then starts the hypervisor and IMA-enabled kernel, which continues the boot process by measuring each successive component before loading it.

The BIOS and boot loader are removed from the trusted computing base by establishing a dynamic root of trust after they are loaded. Care must be taken to ensure that the loaded BIOS and boot loader code are not referenced again—Section 5 explains how this is accomplished. The boot loader merely reads the relevant kernel files into memory so that the secure loader does not need to include support for file systems or storage devices. This use of an untrusted boot loader reduces the complexity of the secure loader, a security-critical component.

After the kiosk boots, it generates a keypair that will allow the phone to encrypt secrets destined for the kiosk. The kiosk includes the public key K in a self-signed certificate $\mathcal{C}$, and extends a PCR with the digest of $\mathcal{C}$. If the phone later decides that the kiosk is trusted, this will imply that the kiosk software will not divulge the private key. As the

0	Shut down Hypervisor/OS
1	Run BIOS and Boot Loader
2	Establish DRTM
3	Run Secure Loader
4	Boot Hypervisor/OS

Figure 5: Reboot sequence

keypair is only for this session, the private key should not be written to stable storage.

4.3.5 Verify the integrity of kiosk software

When the reboot completes, the kiosk alerts the phone and sends it the self-signed certificate of the newly generated public key (Step 9) . In this step the kiosk also provides a signed database of the expected measurements for the chosen configuration. The phone verifies the database's signature using the public key of the trusted third party. The phone then proceeds to verify the integrity of the software loaded in the boot process using the IMA protocol [23] (Steps 10–12). Briefly, the phone challenges the kiosk to produce a quote signed by the TPM in Step 10. Step 11 depicts the creation of this quote. All relevant PCRs are hashed together along with the nonce from Step 10 and signed by the TPM using the private AIK. The inclusion of the nonce prevents the replay of a quote from a previous session, and the cryptographic properties of the hash function prevent an alternative boot sequence from producing the same value for h in Step 11. The kiosk provides the phone with the signed quote and a measurement list describing the boot sequence that produced the final PCR values represented in the quote (Step 12).

To verify that the quote is legitimate (Step 13), the phone first verifies the signature on the quote using the public portion of the AIK obtained in Step 2. The phone then replays the series of PCR extensions described in the measurement list and computes $h' \leftarrow \text{hash}(PCR_0 || \ldots || PCR_N)$. Both h' and the nonce supplied in Step 10 must match the contents of the signed quote.

Finally, the phone verifies that each software component in the measurement list is trusted by referencing the database of known trusted software obtained in Step 9. The only measurement that will not be in the database is the credential $\mathcal{C}$ containing the public key for this session. The phone must, however, ensure that $\mathcal{C}$ is actually included in the measurement list. At the end of this phase (Step 14), the phone trusts the kiosk and is in possession of a public encryption key K generated by the kiosk.

4.3.6 Use kiosk

At this point, the phone encrypts any necessary personal information and sends it to the kiosk (Step 15). The personal information can take many forms depending on the use case. For example, it may be a password for accessing an email account, a credit-card number for making a purchase, or a key for decrypting a user's virtual machine image. If the user selected to resume a personalized VM, the steps shown in Figure 6 will replace Step 15 of Figure 4.

In Step 1 of Figure 6, the mobile device sends to the kiosk a URL where the encrypted VM image can be accessed. The image may reside on a network server as in the ISR model [14], or be carried on the device itself as in the SoulPad model [7]. In Step 2, the kiosk fetches that image. In Step 3, the device sends to the kiosk the key with which to decrypt the image over an encrypted channel. The channel encryption key is the same one generated in Step 8 of Figure 4. Finally, in Step 4 of Figure 6, the kiosk decrypts, loads, and resumes the VM image.

4.3.7 Clean up kiosk

After using the kiosk for some amount of time, the user will indicate that she is finished (Step 16 of Figure 4). At this point, we want to ensure that no personal data can be retrieved from the kiosk after the user walks away. To avoid leaving any traces of data on the kiosk's disk, the user's virtual machine must operate out of an encrypted file system [7]. The only key that the hypervisor must maintain is the decryption key for the VM itself. When the user finishes, the hypervisor suspends the VM, zeroes the memory that was allocated to the VM, and zeroes the memory that stored the VM's decryption key. By this point, we have established that the hypervisor is trustworthy, and we therefore trust it to complete these steps correctly.

4.4 Security Properties

The barcode captured by the phone binds the public portion of the AIK to the TPM in the machine physically in front of the user. The signature on the quote and the cryptographic properties of the hash function bind the observed measurement sequence to the TPM holding the AIK described above. The nonce in the quote binds the quote to this session. From this, the phone can conclude that the kiosk in front of the user did in fact boot the reported software stack while the user was physically present. By referencing all loaded software against a database of trusted software, the phone can then conclude that the kiosk is trustworthy.

4.5 Kiosk Supervisor

The goal of the kiosk owner is to detect when a kiosk is being used inappropriately. In addition to any standard external monitoring mechanisms (e.g., intrusion detection systems), the kiosk supervisor machine may also use the IMA protocol to monitor which software has been loaded on the kiosk. In the scenario where the user uses only kiosk-provided software, the kiosk supervisor can reference all loaded software against a database containing measurements of known-trusted software. In the scenario where the user runs a personalized VM, that VM is unlikely to respond to attestation challenges from the kiosk supervisor for privacy reasons, and therefore the supervisor can only ensure that a trustworthy hypervisor environment is running on the kiosk. This hypervisor environment can, however, be configured to monitor the external behavior of the VM and suspend it should it misbehave.

5. PROTOTYPE IMPLEMENTATION

This section describes our implementation, using commodity components, of a trustworthy kiosk computing system that embodies the design presented in the previous section. Our prototype comprises three parties shown in Figure 1: a mobile device, a kiosk, and a kiosk supervisor. Our mobile device is a Nokia N70 smartphone with a 220 MHz ARM processor and 32 MB of memory, along with GSM/GPRS/EDGE and Bluetooth radios. The smartphone

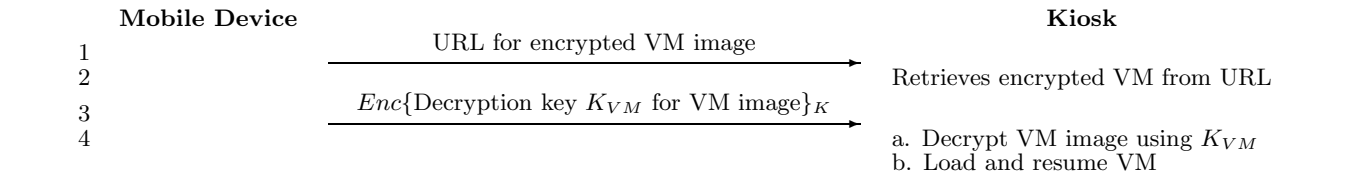

Figure 6: Protocol for resuming a user-specified virtual machine on a kiosk

is a Symbian Series 60 platform, which supports Java Micro Edition (Java ME).

Our kiosk is a desktop PC equipped with a 2.4GHz AMD Secure Virtual Machine-capable x86 processor, 2 GB of memory, 60 GB of disk, an Infineon TPM 1.2, and an Iogear USB Bluetooth adapter. The kiosk runs the Xen hypervisor managed by a virtual machine running Linux. Our kiosk supervisor is a generic Linux PC that runs an existing IMA verifier [23] to periodically request an integrity attestation from the kiosk.

5.1 Mobile Device Software

For the phone we wrote a Java ME application that interacts with the user via the screens shown in Figure 3, and talks to the kiosk over Bluetooth. A wired connection would offer greater security than Bluetooth, but is impractical here because the cables are generally phone-specific. This application includes an IMA verifier written for the Java ME environment. We used the Bouncy Castle [20] library for all cryptographic operations carried out by the IMA verifier, such as replaying PCR extend operations and verifying TPM signatures.

Currently, there is no open-source Java ME implementation for capturing a barcode using the phone's camera, so our prototype omits this functionality. However, the viability of this approach on the Symbian Series 60 platform has been demonstrated in both C++ [17, 24] and Java [5]. In addition to the hash of the AIK certificate, the barcode will also encode the Bluetooth address of the kiosk, which allows the phone to bypass the exceedingly slow device discovery protocol [24].

5.2 Kiosk Software

We added three software components to the kiosk platform: a new kiosk front-end application, an existing IMA attestation server [23], and a modified version of the OSLO secure loader [12].

Kiosk front-end.

The front-end interacts with the phone over Bluetooth to establish the desired software configuration, reboots the kiosk into this configuration, and provides a conduit for the phone to retrieve measurements from the IMA attestation server. The front-end application is written in Java Standard Edition (Java SE), with some help from Perl scripts to manipulate the configuration of the GRUB boot loader.

IMA attestation server.

We employ an unmodified IMA attestation server [1], which listens for requests via TCP. To avoid the additional difficulty of layering TCP on Bluetooth, the phone transmits IMA requests as a Java byte stream to the kiosk front-end, which in turn communicates with the IMA attestation server via loopback TCP.

Secure loader.

Figure 5 outlines the kiosk boot sequence. After rebooting, the BIOS runs the GRUB boot loader, which in turn launches the OSLO secure loader. We use GRUB to load multiboot modules from disk, allowing OSLO to remain simple. OSLO establishes a dynamic root of trust for measurement (DRTM) by invoking skinit, then measures and runs the remaining multiboot modules, in particular the Xen hypervisor and Linux kernel. As described in Section 3, skinit atomically measures the secure loader itself, stores the result in the TPM, and transfers execution to that loader.

We extended OSLO to record, prior to calling skinit, the measurements of itself, the hypervisor, and the kernel in the Advanced Configuration and Power Interface (ACPI) tables maintained in system memory by the BIOS. Standard OSLO does not keep a list of the measurements made by skinit and by OSLO itself. As described in Section 3, such a list is needed by the IMA verifier to replay the measurement sequence. We used the ACPI tables to communicate these measurements to IMA because there is no higher-level communication facility (e.g., a file system) available from within OSLO. Our extensions involved calling a BIOS interrupt routine from 16-bit real-mode x86 assembly code. This routine records measurements in the ACPI tables and is available on all BIOSes that support the TPM. While the ACPI tables are maintained by the BIOS, they may be read directly from system memory by the OS.

As mentioned previously, software that runs after the DRTM is established must never invoke code that has not already been measured into the TPM. In the case of our prototype, nothing may invoke the BIOS or boot loader after OSLO executes skinit because OSLO does not measure the BIOS or boot loader. Our prototype kiosk satisfies this requirement because neither Xen or the paravirtualized version of Linux used in Xen virtual machines ever call back into the BIOS or boot loader, or permit applications running inside a VM to do so.

5.3 Personalized Environments

Our prototype supports personalized computing environments through the use of migrateable Xen virtual machines. The base software stack on our kiosk consists of the Xen 3.0 hypervisor plus a management VM running Fedora Core 4 Linux. The phone first determines if this software stack is trustworthy by carrying out the protocol in Figure 4 through Step 14. The phone then specifies a user VM to run and the kiosk loads and resumes that VM, as per the protocol in Figure 6. Our sample user VM runs Fedora Core 6 Linux. The VM's suspended state is encapsulated in an encrypted

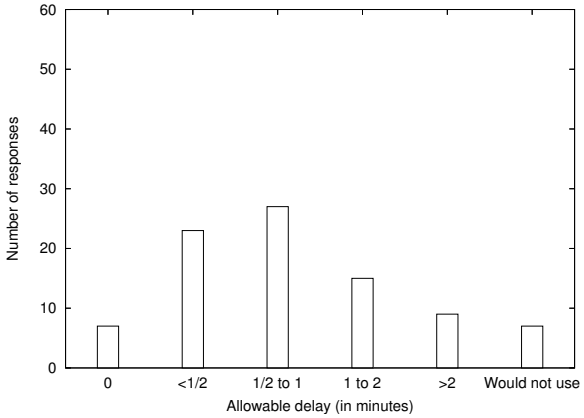

Figure 7: Tolerable delay prior to web surfing

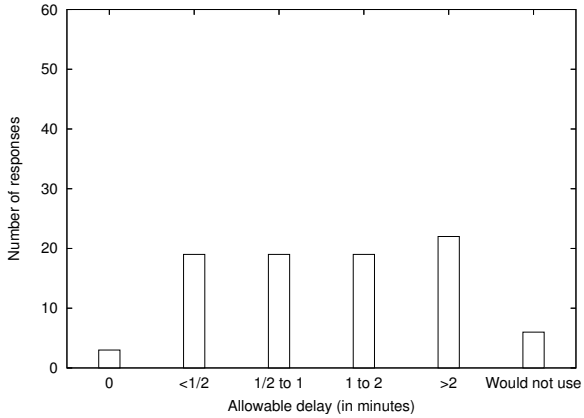

Figure 8: Tolerable delay prior to checking email

compressed archive containing three files: a configuration file used to start the VM, a file containing the VM's main disk image, and a file containing the VM's swap disk image. The size of this archive is 104MB for our sample user VM that uses 1GB of uncompressed disk space. When Xen suspends a VM, it zeroes the memory allocated to that VM, but does not, to our knowledge, zero the memory used to store the VM's decryption key. This minor change would allow Xen to fully implement the cleanup procedure described in Section 4.3.7.

Many refinements are possible on this basic proof of concept [7] [14]. The emphasis of this work has been on resolving the trust issues surrounding the use of personal virtual machines on public hardware.

6. EVALUATION

This section presents an evaluation of user expectations in the kiosk scenario, together with relevant performance characteristics of our prototype system.

6.1 User Expectations

We conducted a survey to determine the extent to which users would tolerate an additional delay prior to using a kiosk, in exchange for an assurance that there is no malware running on the kiosk. We administered the survey via the SurveyMonkey website [2]. The 88 respondents were volunteers solicited from a popular campus-wide discussion board at Carnegie Mellon University.

After giving the respondents a brief description of the threats malware poses on a kiosk and the type of delay incurred by using our system, we asked them the following questions:

1. How long would you be willing to wait if you planned on only surfing the Internet?
2. How long would you be willing to wait if you were going to check your email?
3. How long would you be willing to wait if you were going to make a financial transaction through your bank's website?

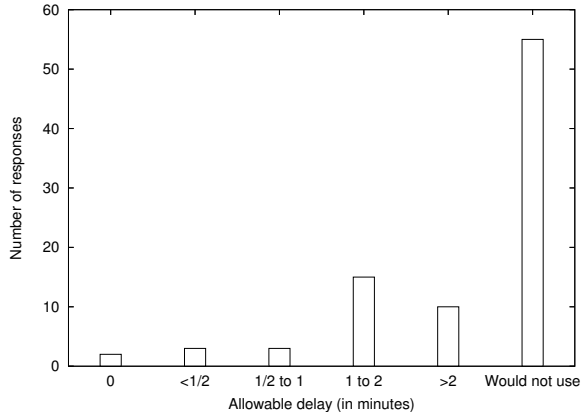

Figure 9: Tolerable delay prior to electronic banking

For each of these questions, the respondents selected one of the following choices:

a. I would not be willing to wait any extra time.
b. 1 to 30 seconds.
c. 31 seconds to 1 minute.
d. 1 to 2 minutes.
e. More than 2 minutes.
f. I would never use a computer at an Internet cafe for this purpose.

The distribution of responses by question are shown in Figures 7–9. The results show that when the intended activity is surfing the web, 58% of respondents would be willing to wait longer than 30 seconds, and 27% would be willing to wait longer than a minute. When the intended activity is email, these percentages increase to 68% and 47%, respectively. Most (63%) of respondents would not be willing to perform bank transactions on a general-purpose kiosk, but of those that would, 74% would be willing to wait longer than a minute.

To estimate the delay that users presently tolerate, we measured the amount of time necessary to log in to a frequently used campus computer cluster. This particular cluster supports the migration of a personalized environment,

consisting of settings and network folders, between machines. As such, the usage model of this cluster closely resembles that of our kiosk. We define the log-in time to be the delay between the time the user clicks "log in" (after entering the password), and the point at which the GUI becomes responsive enough to open the Windows start menu. We measured the log-in time on three Pentium 4 machines running Windows XP Service Pack 2, with clock speeds ranging from 2.8-3.0 GHz and installed memory ranging from 1.0-1.5 GB. The fastest log-in time observed was approximately 90 seconds, and the slowest was 105 seconds. This suggests that the benefits of using a personalized computing kiosk are sufficient for users to tolerate, albeit grudgingly, a rather lengthy initial delay.

6.2 Data Sizes

We measured the sizes of data used by our trust establishment procedure to verify that they are practical in a mobile computing setting. Table 1 shows the sizes of the three main data sets used for integrity verification by the unoptimized prototype described in the previous section.

First, the measurement database includes an entry for each known software component that may run on the kiosk. Each entry contains the SHA-1 hash of the component plus an ASCII string further describing the component. The string contains the name of the file where the component is installed and an indication of whether the component is trusted or untrusted. The Xen and Fedora Core 4 Linux distributions installed on our prototype kiosk generated 20,929 such entries with a total database size of just under 1.3 megabytes and an average entry size of 65 bytes. This database is sent by the kiosk to the phone in Step 9 of the protocol in Figure 4.

Second, the measurement list includes an entry for each unique software component that has been loaded on the kiosk. Each entry contains the SHA-1 hash of the component, the number of the Platform Configuration Register that was extended with this hash value, and an ASCII string holding the name of the file from where the component was loaded. In our prototype, the measurement list sent by the kiosk to the phone in Step 12 of our protocol contained 676 entries, for a total list size of just under 38 kilobytes and an average entry size of 57 bytes. This list grows very slowly over time because measurements are only added to the list when a previously unseen software component is loaded.

Third, the Trusted Platform Module quote includes the contents of the requested Platform Configuration Registers, a digital signature, and a nonce. The TPM quote sent by the kiosk to the phone in Step 12 of our protocol contained 796 bytes. The quote format follows the TPM standard and its size is fixed for a given number of requested PCR values. We always request all 24 PCR values in our prototype.

The above data sizes are already well within the storage capacity of modern smartphones. The Nokia N70 in our prototype system has 32 MB of built-in storage and a socket for adding additional flash memory. Furthermore, there is ample room for reducing the measurement database and list sizes via compression because they are currently composed mostly of uncompressed ASCII text. For example, applying Lempel-Ziv compression yields a measurement database of 630,941 total bytes (46% of the uncompressed size), and a measurement list of 21,377 total bytes (55% of the uncompressed size).

Component	Total Size (bytes)	# of Entries (count)	Average Entry (bytes)
Measurement DB	1,360,640	20,929	65
Measurement List	38,867	676	57
TPM Quote	796	1	796

Table 1: Integrity verification data sizes

Stage	Delay Average (seconds)	Delay Std Dev (seconds)
Xen + Linux Shutdown	25	1.15
OSLO + GRUB Secure Load	2	0.58
Xen + Linux Boot	66	0.58
Measurement DB Transfer	42	3.27
Integrity Verification	23	0.58

Table 2: Trust establishment delays

6.3 Delays

We measured the delays introduced by our trust establishment procedure to determine whether they would be tolerable to users. Table 2 shows the most salient delays incurred by our unoptimized prototype. More specifically, the delays in the table correspond to protocol Steps 8 through 14 in Figure 4. The averages and standard deviations were calculated across three runs of the protocol.

Some of the largest delays were due to the reboot sequence called for by Step 8. Shutting down Xen and Linux took approximately 25 seconds, running the combination of OSLO and GRUB took 2 seconds, and booting Xen and Linux took 66 seconds, for a total reboot latency of 93 seconds.

Table 2 also shows the delay incurred by transferring the measurement database from the kiosk to the phone in Step 9 of our protocol. This transfer took approximately 42 seconds. The last row in Table 2 shows the delay incurred by the integrity verification stage of our protocol. This stage includes everything in Steps 8 through 14 with the exception of the reboot sequence and measurement database transfer discussed above. This stage took approximately 23 seconds, for a total latency without rebooting of 65 seconds. The total delay including rebooting, transferring the measurement database, and verfying software integrity was roughly two and half minutes.

Potential optimizations.

There is a great deal of room for improvement in these delays. With respect to the reboot sequence in Step 8, the kiosk could boot into a clean configuration stored in a static disk image since there is no need for a public kiosk to save user state. This procedure would eliminate the shutdown stage in Table 2 and thus reduce that reboot latency by approximately 25 seconds.

Other parts of the reboot sequence can also be optimized extensively. For example, our prototype uses Fedora Core 4 Linux, a standard software distribution not tailored to the kiosk scenario. Fedora Core includes several dozen software services that by default are started during boot and stopped during shutdown. It should be possible to shave many sec-

onds off the reboot sequence by removing unnecessary services from the sequence.

Nevertheless, even a streamlined reboot sequence may incur a delay that is unacceptable when the user intends to use the kiosk for only a short time. We introduced the reboot sequence in Step 8 of our protocol because we provide only load-time guarantees of integrity. A reboot there offers the greatest degree of security by minimizing the time between software measurement and use. However, it may be acceptable to boot the kiosk into the most likely configuration prior to the user's arrival. If this configuration meets the needs of the user, the phone could skip Steps 8–9 of the protocol. Steps 10–14 would still verify the integrity of all loaded software, but the guarantee provided by these steps would be weaker because the software would have been running for longer.

With respect to the transfer of the measurement database, ongoing advances in short-range wireless networks will greatly reduce the delay of this stage. Our prototype uses the Bluetooth 1.2 standard with a maximum transfer rate of 90 kilobytes per second. Devices are already available that use Bluetooth 2.0 with a maximum transfer rate of 263 kilobytes per second, almost 3 times faster. A future version of Bluetooth is being planned to provide up to 60 megabytes per second, several orders of magnitude faster. These bandwidth improvements will also translate into delay reductions in the integrity verification stage of our protocol, since that stage also transfers a significant amount of data, most notably the measurement list.

Discussion.

Comparing our delay measurements with our survey results, the trust establishment latency of our unoptimized prototype system is already acceptable to a small fraction of our respondents. We believe that the total delay incurred in Steps 8–14 could be reduced to less than one minute by eliminating the kiosk shutdown stage, removing unnecessary software services on the kiosk, and upgrading to Bluetooth 2.0. This delay would then be acceptable to 47% of the survey respondents who want to check their email. The vast majority of respondents were willing to incur some additional delay for increased security, so it seems reasonable to expect that a carefully engineered kiosk could satisfy their needs.

We conclude from our user survey, delay measurements, and data size characterization that it is feasible to use a mobile phone to establish the trustworthiness of a kiosk following the procedure we have proposed. Tailored optimization, along with expected hardware and software advances, will further improve performance and usability.

7. OPEN ISSUES AND FUTURE WORK

This section discusses issues left open by our work to date on trustworthy kiosk computing, and suggests possible approaches to resolving these issues.

Hardware attacks.

While our work focuses on detecting software attacks, hardware attacks are a very real concern in the public kiosk scenario. An attacker may have physical access, and can install a USB sniffer or even replace hardware components such as the network or video card with malicious versions. As detecting such attacks is difficult and tamper-resistant hardware is expensive, we believe that the kiosk should have as many peripherals as possible integrated onto the motherboard, and all I/O devices should be connected without external cables (as is done in laptops), preferably soldered to the motherboard. Simpler physical security measures such as enclosing the kiosk hardware in a locked plastic case can also raise the bar for attackers.

Barcode attacks.

While technically a hardware component, the barcode affixed to the kiosk deserves further consideration. Our system assumes that the barcode correctly identifies the AIK of that kiosk's TPM. However, an attacker with physical access to kiosk A may be able to replace the barcode in a way that escapes the user's notice. The malicious barcode could then identify the AIK of a TPM in kiosk B. If the attacker compromises kiosk A, A could be configured to simply forward input to kiosk B and display B's output on A's screen. B would boot the trusted software stack, and provide an attestation to the user's phone, which would believe that it came from A. As a result, A can monitor the user's actions and what is displayed on the screen. This highlights an important problem with current TPMs: it is difficult for the user to verify the physical location of any TPM. To our knowledge, the barcode is the best solution available to users of commodity hardware.

Runtime attacks.

The measurements stored in the TPM guarantee the integrity of all software *loaded* on the kiosk. However, software may be compromised after it is loaded, e.g., by code-injection attacks. Providing stronger run-time guarantees is a difficult problem and the focus of active research (e.g., [25]). Future run-time attestation solutions can be incorporated into our trust-establishment protocol alongside IMA.

Time of measurement to time of use.

There is a window of time between when the measurement is taken from the TPM and the user begins using the kiosk. If a malicious program is loaded during this time, it may compromise the user's privacy. To limit this vulnerability, the user's phone can periodically request new measurements in the background and alert the user (e.g., by playing a tone) if a malicious program was loaded.

Cleanup with kiosk-provided software.

If the user elects not to use a personalized environment, and instead uses software provided by the kiosk, then cleanup becomes more complicated than the case where a virtual machine is run out of an encrypted file system. We believe that a similar solution can be employed where all writable partitions are configured to use a file system that is encrypted with a key generated for this session, but this approach needs further investigation.

8. RELATED WORK

Previous research has investigated several usage models for kiosk computing. We adopt the model in which the kiosk allows the user to resume a suspended virtual machine (VM), thus providing the user with a personalized computing environment. Internet Suspend/Resume [14] and Soul-

pad [7] describe two different realizations of this model: the former migrates portions of a VM over a network as needed, while the latter resumes a VM from portable storage provided by the user. Neither work thoroughly addressed the security ramifications that are the focus of this paper. Surie et al. [28] propose that the kiosk boot from a USB flash drive provided by the user. The software on this drive may establish a software root of trust prior to booting the kiosk. However, this approach remains susceptible to attacks in which the BIOS has been compromised or the entire environment is booted within a rogue VM monitor. Nishkam et al. [22] use software-only techniques to verify the integrity of the BIOS and check for the presence of a VM monitor. Our approach establishes a hardware root of trust and is thus more secure against both BIOS- and VM monitor-based attacks.

An alternative model is for the kiosk to operate as a thin client, acting as an input/output device for programs that are run elsewhere. Since all computation is performed remotely, the security of this model reduces to the security of the input and output operations performed at the kiosk. Oprea et al. [21] propose that the user perform all input through a trusted PDA, which relays the commands to the remote server via an encrypted tunnel. In addition, Sharp et al. [26] propose that the kiosk only display an obfuscated view of the remote server. The kiosk aids in high-level navigation operations, but all security sensitive input and output operations are performed through the PDA. The drawback to these approaches is that they limit the functionality of the kiosk, whereas our approach does not.

Another approach is to use kiosk-provided software, but limit the amount of information that is revealed to that software. MP-Auth [15] is a protocol that allows a user to authenticate to a remote party (such as a bank's website) without trusting their PC. MP-Auth allows the user to input their password using a trusted mobile phone, and the protocol ensures that the plaintext password is not revealed to any other device. Bump in the Ether [18] uses a trusted mobile phone in conjunction with a TPM to establish a secure tunnel to an application running on a PC. All trusted user input is performed using the phone, or a wireless keyboard that communicates with the PC through the phone. Flicker [16] establishes a dynamic root of trust to verifiably run security-sensitive code in isolation from all other software, including the operating system. Secret data may be stored using the TPM in such a way that it can only be accessed while this security-sensitive code is executing. The challenge of these approaches is that there must be a well-defined distinction between security-sensitive data and standard data. In contrast, we treat all user data as sensitive.

Arbaugh et al. [3] propose an architecture for *secure boot*, in which each layer in the software stack is measured and compared to an expected value before it is loaded. If the software is not as expected, the architecture attempts to recover by obtaining the correct version of the software from either local backup or a trusted third party. Thus a machine will not boot using compromised software. In our scenario, the phone, not the kiosk, ascertains the trustworthiness of loaded software, so it is more efficient to adopt the *trusted boot* approach in which the software is verified after it is loaded.

Previous research considered kiosks in other security-sensitive applications. Sinclair and Smith [27] propose that a phone, in possession of a sensitive private key, temporarily delegate authority to a short-lived keypair known to the kiosk. The trustworthiness of this kiosk is explicitly encoded in a credential signed by the system administrator. In contrast, our system verifies the load-time integrity of the software running on the kiosk, which we feel provides a stronger security guarantee. Furthermore, our system considers the confidentiality of general user data (e.g., a personal VM) in addition to private keys.

Many of the systems described above, including ours, utilize a mobile device under the assumption that it is more trustworthy than the kiosk in question. Balfanz and Felten [4] explore this assumption and maintain that it holds. McCune et al. [19] emphasize that some simple device is necessary to help users determine if a more complex machine can be trusted. They envision a minimal device that indicates yes/no trustworthiness via a green/red light. Such a minimal device, if realized, could be useful in our scenario, but a phone enables more complex operations, which are needed if the user is to select and resume custom environments.

9. CONCLUSION

We have presented the design of a system in which a user's mobile device serves as a vehicle for establishing trust in a public computing kiosk by verifying the integrity of all software loaded on that kiosk. This procedure leverages several emerging security technologies, namely the Trusted Platform Module, the Integrity Measurement Architecture, and new x86 support for establishing a dynamic root of trust. Our system balances the desire of the user to maintain data confidentiality against the desire of the kiosk owner to prevent misuse of the kiosk. We have demonstrated the viability of our approach by implementing our system on commodity hardware. The delay incurred by the trust establishment protocol in our prototype is close enough to the range of delays reported as tolerable by users that we feel moderate engineering effort would result in a useable system.

In this paper we have focused on allowing the user to personalize a kiosk by running her own virtual machine there. However, our work is generally applicable to establishing trust on public computing devices before revealing any sensitive information to those devices.

Acknowledgments

We thank our shepherd Maria Ebling and the anonymous reviewers for comments and suggestions that helped to improve the paper. We thank Jonathan McCune and Bryan Parno for their advice regarding various technical aspects of our prototype.

10. REFERENCES

[1] Integrity Measurement Architecture Implementation. http://sourceforge.net/projects/linux-ima.

[2] SurveyMonkey. http://surveymonkey.com/.

[3] W. Arbaugh, D. Farber, and J. Smith. A Secure and Reliable Bootstrap Architecture. In *Proc. of the IEEE Symposium on Security and Privacy*, 1997.

[4] D. Balfanz and E. Felten. Hand-held computers can be better smart cards. In *Proc. of the USENIX Security Symposium*, 1999.

[5] L. Bauer, S. Garriss, J. M. McCune, M. K. Reiter, J. Rouse, and P. Rutenbar. Device-enabled authorization in the Grey system. In *Information Security: 8th International Conference, ISC 2005*, 2005.

[6] S. Brands. Untraceable off-line cash in wallet with observers (extended abstract). In *Advances in Cryptology – CRYPTO '93*, Lecture Notes in Computer Science, 1993.

[7] R. Cáceres, C. Carter, C. Narayanaswami, and M. T. Raghunath. Reincarnating PCs with Portable SoulPads. In *Proc. of the ACM/USENIX Conference on Mobile Computing Systems, Applications, and Services*, 2005.

[8] Advanced Micro Devices. Secure Virtual Machine Technology. http://www.amd.com/.

[9] S. Garriss, R. Cáceres, S. Berger, R. Sailer, L. van Doorn, and X. Zhang. Towards Trustworthy Kiosk Computing. In *Proc. of the IEEE Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2007.

[10] Trusted Computing Group. Trusted Platform Module. https://www.trustedcomputinggroup.org/.

[11] Intel. Trusted Execution Technology. http://www.intel.com/technology/security/.

[12] B. Kauer. OSLO: Improving the Security of Trusted Computing. In *Proc. of the USENIX Security Symposium*, 2007.

[13] S. T. King, P. M. Chen, Y. M. Wang, C. Verbowski, H. J. Wang, and J. R. Lorch. SubVirt: Implementing Malware with Virtual Machines. In *Proc. of the IEEE Symposium on Security and Privacy*, 2006.

[14] M. Kozuch and M. Satyanarayanan. Internet Suspend/Resume. In *Proc. of the IEEE Workshop on Mobile Computing Systems and Applications*, 2002.

[15] P. C. van Oorschot M. Mannan. Using a Personal Device to Strengthen Password Authentication from an Untrusted Computer. In *Proc. of the International Conference on Financial Cryptography and Data Security*, 2007.

[16] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An Execution Infrastructure for TCB Minimization. In *Proc. of the European Conference on Computer Systems (EuroSys)*, 2008.

[17] J. M. McCune, A. Perrig, and M. K. Reiter. Seeing is Believing: Using Camera Phones for Human-Verifiable Authentication. In *Proc. of the IEEE Symposium on Security and Privacy*, 2005.

[18] J. M. McCune, A. Perrig, and M. K. Reiter. Bump in the Ether: A Framework for Securing Sensitive User Input. In *Proc. of the USENIX Annual Technical Conference*, 2006.

[19] J. M. McCune, A. Perrig, A. Seshadri, and L. van Doorn. Turtles all the way down: Research challenges in user-based attestation. In *Usenix Workshop on Hot Topics in Security (HotSec'07)*, 2007.

[20] Legion of the Bouncy Castle. Bouncy Castle Lightweight Cryptography API. http://www.bouncycastle.org/.

[21] A. Oprea, D. Balfanz, G. Durfee, and D. Smetters. Securing a remote terminal application with a mobile trusted device. In *Proc. of the Annual Computer Security Applications Conference*, 2004.

[22] N. Ravi, C. Narayanaswami, M. Raghunath, and M. Rosu. Towards Securing Pocket Hard Drives and Portable Personalities. *IEEE Pervasive Computing*, 6(4), 2007.

[23] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Proc. of the USENIX Security Symposium*, 2004.

[24] D. Scott, R. Sharp, A. Madhavapeddy, and E. Upton. Using visual tags to bypass Bluetooth device discovery. *Mobile Comp. and Comm. Review*, 1(2), January 2005.

[25] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla. Pioneer: Verifying Integrity and Guaranteeing Execution of Code on Legacy Platforms. In *Proc. of the ACM Symposium on Operating Systems Principles*, 2005.

[26] R. Sharp, J. Scott, and A. Beresford. Secure Mobile Computing via Public Terminals. In *Proc. of the International Conference on Pervasive Computing*, 2006.

[27] S. Sinclair and S. Smith. PorKI: Making User PKI Safe on Machines of Heterogeneous Trustworthiness. 2005.

[28] A. Surie, A. Perrig, M. Satyanarayanan, and D. Farber. Rapid Trust Establishment for Pervasive Personal Computing. *IEEE Pervasive Computing*, 6(4), 2007.