RC24169 (W0702-003) February 2, 2007
Computer Science

IBM Research Report

Trustworthy Personalized Computing on Public Kiosks

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

[1]Carnegie Mellon University
Pittsburgh, PA

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

[3]AMD
Austin, TX

Trustworthy Personalized Computing on Public Kiosks

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

Carnegie Mellon University[1] IBM T.J. Watson Research Center[2] AMD[3]
Pittsburgh, PA, USA Hawthorne, NY, USA Austin, TX, USA

Abstract

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 revealing personal information to that kiosk. We have designed and implemented a protocol by which the mobile device determines the identity and integrity of the software loaded on the kiosk. A similar protocol allows a kiosk owner to verify that the kiosk has loaded only approved software. Our system combines a number of emerging security technologies, including the Trusted Platform Module, the Integrity Measurement Architecture, and new support in x86 processors for establishing a dynamic root of trust. We focus on allowing the user to personalize the computing environment on the kiosk by running his own virtual machine.

1 Introduction

Public computing *kiosks*, such as a rental computer at an Internet café, have become commonplace and their numbers are likely to increase as computing finds its way into more people's lives. Kiosks are attractive because they free people from the need to carry a fully featured computing device such as a laptop computer. A particularly compelling service that a kiosk could provide is to allow a user to resume a personalized computing environment that includes her choice of software and data. However, a problem with current kiosks is that the user must assume that a kiosk is performing only its intended function, or more specifically, that it has not been compromised by an attacker. A compromised kiosk could harm the user in many ways, for example, by stealing private data. Similarly, the owner of a kiosk wants to ensure that the kiosk is not used to perform malicious acts for which he may be liable.

This paper presents a system in which a user leverages a lightweight personal mobile device, such as a smartphone, to gain a degree of trust in a kiosk prior to using it. In the context of computer systems, trust is the expectation that a system will faithfully perform its intended purpose. We refer to a kiosk as trustworthy if we can verify the identity and integrity of the software loaded on that kiosk. Our system aims to prevent software attacks on the kiosk; detecting hardware modifications is an open problem. Software attacks such as keystroke logging are far more common and an important threat in their own right. We assume that the personal device is a priori trustworthy. Securing mobile devices is itself an important research area, but we do not address it in this work except to point out that some of the trust establishment procedures we apply to kiosks may also apply to mobile devices.

In this context we have designed a protocol by which the mobile device establishes that the kiosk has loaded only trustworthy software. Only after this protocol has successfully completed will the user reveal

[*]This work was done during a graduate student internship at IBM T.J. Watson Research Center.

personal information, such as credit-card data, to the kiosk. Our system also uses a similar protocol by which a supervisor machine, acting on behalf of the kiosk owner, verifies that the kiosk has loaded only approved software. If unapproved software is found, the owner can take action to disable the kiosk by, for example, removing it from the network.

Our system brings together a number of emerging security technologies. We utilize new x86 processor support for establishing a dynamic root of trust on commodity computing platforms that incorporate AMD's Secure Virtual Machine Technology [10] or Intel's Trusted Execution Technology [15]. In addition, we use the Trusted Platform Module (TPM) [14] together with the Integrity Measurement Architecture (IMA) [24] to provide both user and owner with proof that only trustworthy software has been loaded on the kiosk.

We focus on using virtual machine technology to allow the user to run personal software on the kiosk. Running her own virtual machine on a kiosk provides the user with a complete and personalized computing environment, much as if she was using her own physical machine. SoulPad [9] and Internet Suspend/Resume (ISR) [18] are two mobility solutions based on running users' virtual machines on stationary host computers such as kiosks. The work presented here resolves important trust issues left open by those previous approaches. Namely, our use of integrity attestation eliminates blind trust in any software component on the host PC, including the BIOS. Our use of a hardware root of trust also prevents the otherwise difficult to detect attack where a rogue virtual machine monitor runs below an unsuspecting operating system [17]. Additionally, the virtual machine monitor provides the kiosk owner with a means to detect and isolate user virtual machines that are performing illicit activity.

We have implemented our trust establishment procedures along with the overall system to demonstrate the viability of our solution. Our prototype, depicted in Figure 1, uses a commercially-available smartphone as the personal device, a commodity PC as the kiosk, and a Bluetooth wireless link to communicate between them. We use a second PC as the supervisor machine connected to the kiosk via the wired Internet.

The main contribution of this work is the experimental demonstration of a system for trustworthy kiosk computing that combines the security and virtual machine technologies mentioned above. In an earlier position paper [11], we presented a preliminary design and implementation, and identified a number of open issues. This paper improves on that earlier work on several fronts. One, we prevent a kiosk-in-the-middle attack by cryptographically binding the message that attests software integrity to the particular kiosk in front of the user. Two, we create a secure channel between the device and the kiosk by exchanging an encryption key generated for this session. Three, we add the ability to securely resume a user-specified virtual machine on the kiosk by having the device provide the kiosk the location and decryption key of the virtual machine image.

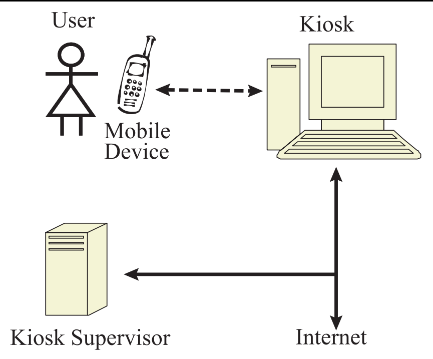


Figure 1: Kiosk computing scenario

The rest of this paper is organized as follows. Section 2 gives an overview of the experience of using our trustworthy kiosk computing system. Section 3 provides some technological foundations necessary to understand the rest of the paper. Sections 4 and 5 present the design and implementation of our trust establishment procedures. Section 6 discusses open issues and future work. Finally, Section 7 summarizes related work.

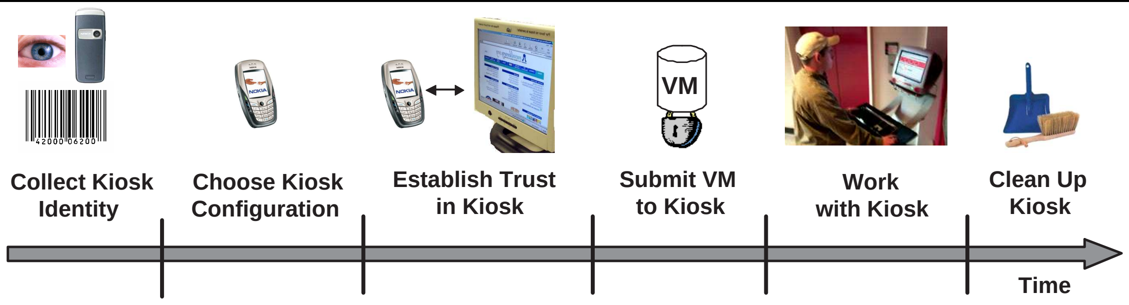


Figure 2: Timeline for secure kiosk use

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 put an undue burden on the user while providing important security functionality. Later sections of this paper will detail what the system does behind the scenes during this interaction.

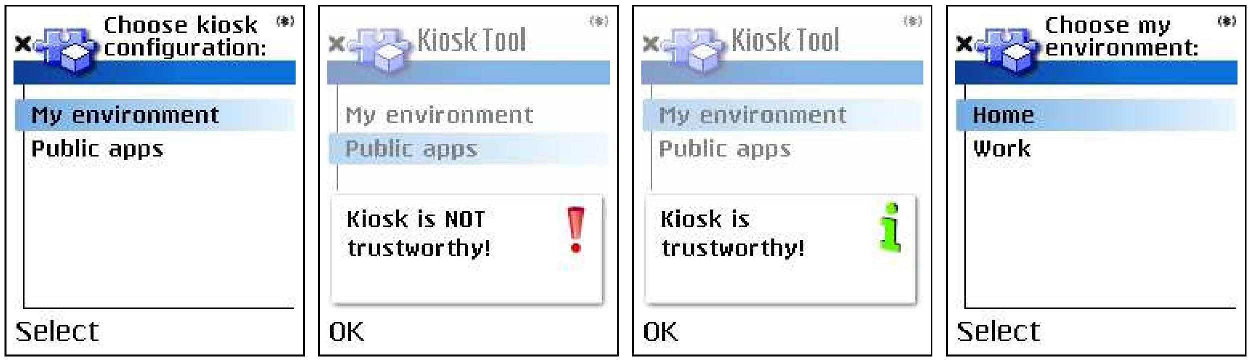
Figure 2 shows a timeline of the steps a user follows after stepping up to a kiosk that she intends to use. First, she makes the identity of the kiosk known to her mobile phone. We propose for kiosk owners to display a numerical identifier on the outside of the kiosk in barcode form. The user captures the contents of the barcode using the digital camera available on modern phones [19].

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.

Fifth, the user proceeds to work on the kiosk. Finally, the user disconnects from the kiosk, which triggers cleanup operations to make the kiosk ready for the next user.



Kiosk is NOT trustworthy!

Kiosk is trustworthy!

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

3 Background

Trusted Platform Module (TPM): The TPM [14] is a hardware component that is increasingly available in personal computers and servers. 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 loaded software components in a set of Platform Configuration Registers (PCRs). PCRs are initialized at boot time 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 back in the PCR. The cryptographic properties of this operation state that it is infeasible to reach the same PCR state through different sequences of inputs. A single PCR can thus store an aggregate representation of an arbitrary sequence of software components. 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., spyware or a keyboard logger), but the TPM cannot detect compromises, such as buffer overflow attacks, that occur after software is loaded. Providing strong run-time guarantees is challenging, and is an area of active research (cf. [27]).

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 is signed by a certification authority. An AIK certificate thus provides a binding between a public key and a certified-legitimate TPM. For privacy reasons, a TPM may have multiple AIKs, but one is sufficient for the kiosk in this paper. The reliance on a certification authority is a commonly perceived weakness of current TPM implementations. Direct Anonymous Attestation (DAA) [8] obviates this reliance and has been adopted by the relevant standards body, but is not available in current TPM implementations. Our system may incorporate DAA as it becomes available.

Integrity Measurement Architecture (IMA): The TPM may be used to achieve *trusted boot*, where measurements stored in PCRs are used to verify that the loaded software stack meets expectations. IMA [24] extends trusted boot by additionally measuring 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 *challenger* (also called a *verifier*) to challenge the integrity of an IMA platform. The challenger 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. The IMA attestation protocol is described in Section 4.3.5 within the context of our trust establishment protocol. The measurement database used by the verifier must be maintained and distributed by a trusted third party, such as a software vendor, who verifies the trustworthiness of the software identified in the database. In our scenario, the database need only include software found in specific kiosk configurations.

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 an init state (which disables interrupts and guarantees atomicity). 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.

4 System Design

Each of the above technologies offer distinct benefits in a kiosk computing scenario and are available on commodity hardware. The TPM provides a root of trust that is implemented in hardware and is thus resilient to many common software attacks. IMA allows the kiosk to use the functionality of the TPM to prove to the user's phone that the kiosk has loaded a particular software stack. A difficulty in using IMA is that the verifier (the phone in our scenario) must have a database against which it can reference all software loaded by the kiosk—our decision to establish a dynamic root of trust is motivated by this problem.

In the general case, this database would need to include the BIOS, the operating system or hypervisor, and any applications that the computer may run. Constructing a database a priori that would be sufficient for establishing trust in any given general purpose computer would clearly be a daunting task. However, an application-specific kiosk (such as an airline check-in terminal) will have dramatically fewer applications and only a few operating system versions. If the user provides a personalized computing environment, the database need not include any applications at all. Thus, we believe it is reasonable to expect that the database contain any software the kiosk will load modulo the BIOS. Despite being a small and relatively stable, the BIOS poses a unique challenge in that it varies widely between machines. Rather than include the BIOS of each potential kiosk in our measurement database, we remove the BIOS from the trusted computing base by establishing a dynamic root of trust after the BIOS is loaded. BIOS measurements may therefore be omitted from the database entirely.

The phone will likely delegate the creation of the database to a trusted third party, who will sign the database and any updates to guarantee their integrity. The phone could periodically contact the trusted third party to obtain updates. Alternatively, the kiosk could present the phone with a database signed by the trusted third party that includes the software installed on the kiosk. This would allow the phone to obtain measurements as needed, but without requiring the trusted third party to be online.

4.1 System Components

As shown in Figure 1, our system consists of a user carrying a mobile device, a kiosk, and a kiosk supervisor. The mobile device runs an application that aids the user in ascertaining the trustworthiness of the kiosk. This application incorporates an IMA verifier. The kiosk is a PC equipped with a DRTM-enabled processor and a TPM. If the kiosk supports personalized computing environments, it will run a hypervisor, otherwise a standard operating system will suffice. The kiosk will additionally run an IMA attestation server and a thin kiosk front-end for talking to the mobile device. The kiosk supervisor may be any platform capable of running an IMA verifier.

4.2 Goals, Assumptions, and Threat Model

Goals We aim to address the concerns of two parties: the user and the kiosk owner. The user's goal is to protect the confidentiality of her data. This includes ensuring that the data is transmitted to the kiosk in secrecy, that the kiosk software will not reveal the data, and that the data cannot be extracted from the kiosk after the user has left. The goal of the kiosk owner is to detect any misuse of the kiosk (e.g., launching a denial of service attack) so that appropriate action may be taken.

Assumptions Our trusted computing base includes the hardware and software on the mobile device as well as the hardware on the kiosk. Verifying the hardware integrity of a platform is an open problem, and as a result, we must assume that the hardware on the kiosk has not been subjected to physical tampering.

This condition implies that we also assume that the barcode affixed to the kiosk has not been modified and properly encodes the certificate corresponding to the AIK in use by the TPM of the kiosk. Capturing a barcode is an attractive method of obtaining the identity of the kiosk because of the simplicity of the user's action, but it is secure only if the integrity of the barcode is guaranteed. Tamper-evident barcodes (e.g., barcodes etched in glass) will mitigate the severity of this problem, but users must remain diligent. Alternative approaches to obtaining the public AIK of the kiosk's TPM are discussed in Section 6.1.

Furthermore, we also assume that the phone is in possession of the public key of a trusted third party. This third party may then certify the public keys used to sign any measurement databases and AIK credentials that the phone receives.

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 evesdrop 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 attacker may not modify the hardware of the kiosk in any way. We note that this definition considers the kiosk owner an attacker as well.

The kiosk owner is concerned with attackers who have physical access to the kiosk, may run arbitrary programs on the kiosk, and may boot the kiosk from a portable storage device.

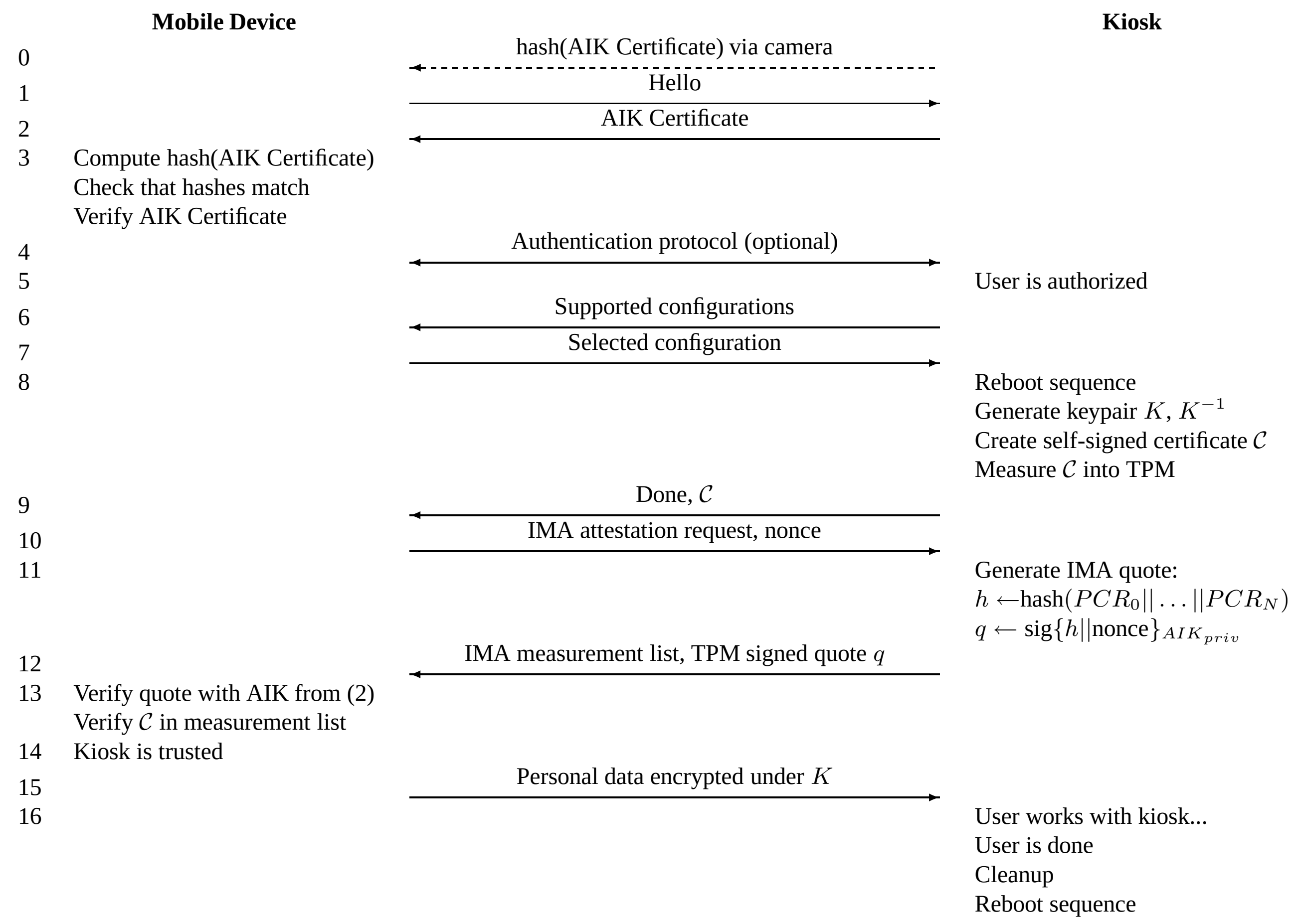

Mobile Device
Kiosk
0 hash(AIK Certificate) via camera
1 Hello
2 AIK Certificate
3 Compute hash(AIK Certificate)
Check that hashes match
Verify AIK Certificate
4 Authentication protocol (optional)
5 User is authorized
6 Supported configurations
7 Selected configuration
8 Reboot sequence
Generate keypair K, K^{-1}
Create self-signed certificate $\mathcal{C}$
Measure $\mathcal{C}$ into TPM
9 Done, $\mathcal{C}$
10 IMA attestation request, nonce
11 Generate IMA quote:
$h \leftarrow$hash$(PCR_0|| \dots ||PCR_N)$
$q \leftarrow$ sig$\{h||$nonce$\}_{AIK_{priv}}$
12 IMA measurement list, TPM signed quote q
13 Verify quote with AIK from (2)
Verify $\mathcal{C}$ in measurement list
14 Kiosk is trusted
15 Personal data encrypted under K
16 User works with kiosk...
User is done
Cleanup
Reboot sequence

Figure 4: Trust-establishment protocol between mobile device and kiosk

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. [19] in which the visual channel provided by the phone's camera is used to securely obtain the public key of the kiosk. First, the user photographs a barcode affixed to the kiosk (Step 0). This barcode encodes the hash of the Attestation Identity Key (AIK) certificate of the kiosk. The AIK certificate includes the public AIK, which is the key that the kiosk's TPM will use to sign integrity measurements later in the protocol.

After capturing the barcode, the phone will initiate the protocol with the kiosk (Step 1). The kiosk then transmits its AIK certificate over the wireless channel to the phone, who compares the hash of the certificate against the one obtained from the barcode.

Having checked that the AIK certificate matches the barcode, the phone will then verify the certificate's signature. If the certification authority that signed the certificate is unknown to the phone, the phone must first gain trust in the certification authority's public key through some form of public-key infrastructure. This step is intrinsic to any application based on current TPM implementations. Verifying the AIK certificate allows the phone to conclude that the AIK was generated by a certified-legitimate TPM. At the conclusion of this phase, the phone knows the kiosk's TPM is legitimate. It also knows the public AIK that the kiosk's TPM will use to sign integrity measurements.

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 allowed to use the kiosk.

4.3.3 Select desired configuration

In Step 6, the kiosk informs the phone of the valid configurations that it may boot. The objective is for the phone to select a configuration that allows the user to perform her intended action and consists of software that the phone knows to be trusted. If these configurations encompass different use cases (e.g., the kiosk may boot a hypervisor or a standard set of public applications), the phone must query the user to determine the desired use case. Other configuration differences, e.g., particular Linux kernel versions, may be selected by the phone without user interaction.

If the phone's database of trusted software does not contain measurements for all of the software included in the selected kiosk configuration, the phone may attempt to obtain an updated database signed by a trusted third party via the cellular network or directly from the kiosk itself. After Step 7, the phone and the kiosk have agreed on a configuration that, if booted correctly, the phone will trust.

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 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/or OS kernel files. The secure loader then starts the hypervisor and/or 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 was accomplished. The boot loader is used merely to read the relevant kernel files into memory; including file system and storage device support in the secure loader would significantly increase the size of this security-critical component.

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

Figure 5: Reboot sequence

After the kiosk boots, it generates a new keypair that will allow the phone to encrypt secrets destined for the kiosk. The kiosk includes the public key in a self signed certificate, and extends a PCR with the digest of this certificate. If the phone eventually determines that the kiosk is trusted, this will imply that the kiosk software will not divulge the private key. As the keypair is generated for this session only, the private key need not be written to stable storage.

For some applications, the delay incurred by rebooting the kiosk may be unacceptable. This delay can be avoided by mandating a particular configuration and booting the kiosk before the user arrives. Section 6 discusses the security implications of this approach.

At the conclusion of this phase, the kiosk claims to have booted the configuration agreed upon in the previous phase and generated a new keypair for this session, but the phone has not yet validated this claim.

4.3.5 Verify the integrity of kiosk software

When the reboot is complete, the kiosk will alert the phone and include the self-signed certificate of the newly-generated public key (Step 9) . The phone verifies the integrity of the software loaded in the boot process using the IMA protocol [24] (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 and signed along with the nonce from Step 10 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, 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 $h' \leftarrow$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 a database of known trusted software. 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.

The barcode captured in Step 0 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 physically 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 conclude that the kiosk is trustworthy (Step 14). At the end of this phase, the phone trusts the kiosk and is in possession of a public encryption key K generated by the kiosk.

4.3.6 Use kiosk and cleanup

At this point, the phone encrypts any necessary secret information and sends it to the kiosk. This information may be application-specific, e.g., payment methods, or it may include instructions for resuming a user's suspended virtual machine. The latter case is discussed further in Section 4.4. After using the kiosk for some amount of time, the user will indicate that she is finished (Step 16).

At this point, we want to ensure that no personal data may be retrieved from the kiosk after the user walks away. This personal data may reside in memory or on the kiosk's disk. As it is difficult to ensure that all state has been purged from memory, we require the kiosk to reboot. The kiosk's operating system must be configured to zero memory as part of its shutdown process. The user may use visual indicators to gain assurance that the kiosk rebooted, but in scenarios with strict security requirements, the user could wait for the reboot to complete and verify this with IMA.

In the case where the user uses a personalized computing environment, this environment can operate out of an encrypted file system [9]. Since we trust the software on the kiosk, we trust that it will properly maintain the key, i.e., it will not disclose it to a third party or write the key to disk. When the kiosk reboots, the decryption key will be lost and along with it the contents of the encrypted file system. If the user elects not to use a personalized environment, and instead uses software provided by the kiosk, then cleanup is more complicated. We believe that a similar solution can be employed where all writeable partitions are configured to use a file system that is encrypted with a key generated for this session. We have not, however, fully investigated this solution in the context of using kiosk-provided software.

At the end of this phase, the user can conclude that the kiosk no longer possesses her personal data.

4.4 Personalized Computing Environments

We use virtual machine (VM) technology to support the important case where the user wishes to run a personalized computing environment on the kiosk. Internet Suspend/Resume (ISR) [18] and SoulPad [9] have shown how VMs can be used to run complete personal computing environments on kiosks. This computing model is appealing because the user maintains control over the choice of operating system, applications, settings, and data inside her personal VM.

However, both the ISR and SoulPad efforts left unresolved the trust issues that are the focus of this work. With ISR, the user submits her personal virtual machine to run in a complete hypervisor environment that is already running in the kiosk. The Trusted Computing Base (TCB) in the ISR prototype thus includes the BIOS, the boot loader, the Linux host OS, and the VMware Workstation virtual machine monitor. With SoulPad the host boots directly from the SoulPad device, and therefore the TCB is limited to the BIOS. However, a malicious hypervisor running on the kiosk could fool the user into thinking she is booting her SoulPad environment on raw hardware when in fact it is booting inside a virtual machine vulnerable to snooping from the hypervisor.

The trust establishment procedure presented earlier addresses all these problems by using trusted computing hardware to remove the BIOS and boot loader from the TCB, and to present the user proof of the identity and integrity of all other software loaded on the kiosk. We thus use the protocol described in the previous section to verify that a kiosk is running a trustworthy hypervisor environment before a user allows a personal VM to run on the kiosk. In this case, the personal data revealed to the kiosk takes the form of a suspended virtual machine image. Figure 6 shows the specific protocol steps we use to securely resume a user-specified virtual machine on the kiosk. These steps take the place of Step 15 in 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, or be carried on the device

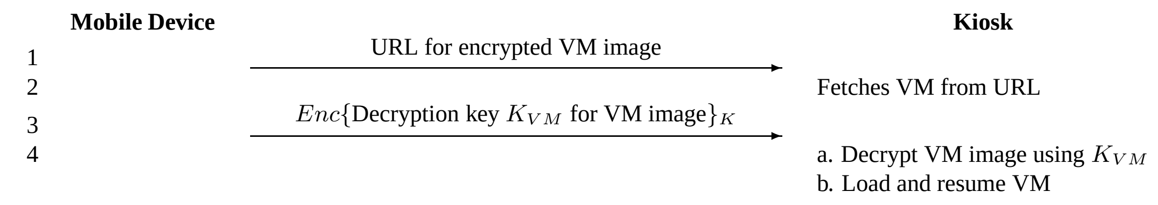

Mobile Device
Kiosk
1
URL for encrypted VM image
2
Fetches VM from URL
3
$Enc\{$Decryption key K_{VM} for VM image$\}_K$
4
a. Decrypt VM image using K_{VM}
b. Load and resume VM

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

itself as in the SoulPad model. In Step 2, the kiosk fetches that image. In Step 3, the device sends the kiosk the key with which to decrypt the image. This decryption key is itself sent encrypted to protect it from eavesdroppers on the wireless channel between the device and the kiosk. The channel encryption key is the same one generated in Step 8 of the main trust establishment protocol shown in Figure 4. Finally, in Step 4 of Figure 6, the kiosk decrypts, loads, and resumes the VM image.

4.5 Kiosk Supervisor

As stated previously, 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 request quotes from the kiosk's TPM 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 measurementes 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. From the standpoint of the kiosk owner, this approach is superior to one in which the user supplies the entire software stack (e.g., Soulpad) because the hypervisor can be configured to limit or prevent certain types of activities that the owner deems inappropriate.

All monitoring activity must, however, be performed without the kiosk owner interactively logging on to the kiosk. Allowing the kiosk owner to login as root while the kiosk is in use would enable the owner to obtain the confidential data of the user by, e.g., reading directly from physical memory. The kiosk boot sequence should therefore be configured to disable remote interactive login. As configuration files are measured by IMA, the user's phone can ensure that such logins are indeed disabled as part of the trust establishment protocol. When the user leaves, the kiosk can boot an alternative "idle" boot sequence that allows the kiosk owner to log in and perform routine maintenance. If the kiosk fails to revert to that idle configuration, the owner can gain physical access to the the kiosk and reboot it manually.

5 Prototype Implementation

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 GSM/GPRS and Bluetooth wireless connectivity. The smartphone is a Symbian Series 60 platform, which supports Java 2 Micro Edition (J2ME). Our kiosk is a desktop PC equipped with an AMD Secure Virtual Machine-capable processor, an Infineon TPM 1.2, and an Iogear USB Bluetooth adapter. The kiosk runs the Xen hypervisor managed by a virtual machine running

Linux. We note that Xen currently does not zero memory prior to a reboot as is required in Section 4.3.6. Such functionality is present in Knoppix distributions of Linux, so it should be feasible to extend Xen in this manner. Our kiosk supervisor is a generic Linux PC.

The rest of this section describes the software we added to the mobile device and kiosk to carry out the trust establishment protocol shown in Figure 4. The kiosk supervisor simply runs an existing IMA verifier [24] to periodically request an integrity attestation from the kiosk.

5.1 Mobile Device Software

For the phone we wrote a J2ME application that interacts with the user and talks to the kiosk over Bluetooth. This application includes an IMA verifier written for the J2ME environment. We used the Bouncy Castle [22] library for all cryptographic operations carried out by the IMA verifier, such as replaying PCR extend operations and verifying TPM signatures. Finally, we pre-loaded measurements of all the software expected to run on our prototype kiosk, including Xen, Linux, and the additional components described below. Stored as ascii text, this database is approximately 1.3MB. The N70 has 32MB of built-in storage and a socket for adding additional flash memory. Since the phone must only store measurements for a handful of kiosk configurations, we do not anticipate that the phone's storage capacity will hinder our ability to maintain a measurement database.

Currently, there is no open-source J2ME implementation for capturing a barcode using the phone's camera, so our prototype does not include this functionality. However, the viability of this approach on the Symbian Series 60 platform has been demonstrated in both C++ [19, 26] and Java [5].

A common difficulty with implementing applications that use Bluetooth is that Bluetooth device discovery is exceedingly slow and does not provide any means to link the device that was discovered with the device the user intends to connect to. Scott et al. [26] address this problem by encoding the Bluetooth address of a device in a barcode that may be captured by the phone's camera, thus obviating the need for device discovery. Current C++ implementations of the barcode recognizer have sufficient resolution to encode a Bluetooth address in addition to the hash of the AIK certificate previously described in Section 4.3.1.

5.2 Kiosk Software

We added three software components to the kiosk platform: a new kiosk front-end application, an existing IMA attestation server [24], and a modified version of the OSLO secure loader [16].

Kiosk front-end: The front-end interacts with the phone over Bluetooth to establish the desired software configuration, reboots the machine into this configuration, and provides a conduit for the mobile phone to retrieve measurements from the IMA attestation server. The front-end application is written in Java 2 Standard Edition (J2SE), with some help from Perl scripts to manipulate the configuration of the GRUB boot loader.

IMA attestation server: We employ an unmodified open-source 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 to the kiosk front-end, which in turn communicates with the IMA attestation server via loop-back 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. OSLO establishes a dynamic root of trust for measurement (DRTM) by invoking skinit, then measures and runs the Xen hypervisor and the 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 the measurements of itself, the hypervisor, and the kernel in the Advanced Configuration and Power Interface (ACPI) table 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 table 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 table and is available on all BIOSes that support the TPM. While the ACPI table is maintained by the BIOS, they may be read directly from system memory by the OS.

There is an important subtlety in the use of a DRTM: 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 must invoke the BIOS or boot loader after OSLO executes skinit because OSLO does not measure the BIOS or boot loader. Our 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. We also had to be careful to update the ACPI table *prior* to executing skinit because updating the ACPI table involved calling the BIOS, as described above. As a result, we had to calculate the SHA1 hash of multiboot modules twice, once to store in the ACPI table from outside the secure loader, and once to extend PCRs from inside the secure loader.

5.3 Personalized Computing Environments

Our prototype supports personalized computing environments through the use of migrateable Xen virtual machines (VMs). The base software environment on our kiosk consists of the Xen hypervisor plus a management VM running Linux. The mobile phone first determines if this environment is trustworthy by carrying out the protocol in Figure 4 through Step 14. The phone then specifies to the management VM a user VM to run, as per the protocol in Figure 6.

Our sample user VM runs the Fedora Core 6 Linux distribution. The VM's suspended state consists of three files: a configuration file that is used to start or resume it, a file containing its main disk image, and a file containing its swap disk image. The configuration file specifies that the two disk image files be used in place of raw disk partitions. Having the user VM mount file systems and swap space directly from these image files allows the VM to be easily migrated between Xen systems. For added portability, the three files are encapsulated in an encrypted compressed archive. The size of this archive is 104MB for our sample user VM that uses 1GB of uncompressed disk space.

Upon receiving from the phone the URL and decryption key for the archive containing a suspended VM, the management VM fetches the archive, decrypts it, and uncompresses it. Finally, the management VM resumes the user VM, thus making available to the user a complete personal computing environment, including an operating system, applications, settings, and data. Many performance improvements are possible on this basic proof of concept. The emphasis of this work has been on resolving the trust issues surrounding the use of personal virtual machines on public hardware.

6 Discussion

6.1 Kiosk-in-the-Middle Attack

The kiosk in front of the user, or local kiosk, could run malicious software that relays data between the mobile device and a second, remote kiosk. The remote kiosk could run and attest to the intended software stack, thus fooling the device into trusting the local kiosk. The local kiosk could then snoop on and misuse personal data. This problem is similar to the chess grandmaster problem [2]. To prevent this attack we need to ensure that the TPM quote was signed by the AIK of the TPM inside the local kiosk.

As described in Section 4.3.1, our solution to this problem is to capture a barcode that encodes the SHA1 hash of the AIK using the camera of the mobile device [19]. The disadvantage of this approach is that it may not be difficult to fool the user into capturing a falsified barcode. Here we explore alternate solutions.

A user could carry a portable TPM with a known AIK. Before initiating the trust-establishment protocol, the user could attach the TPM to the kiosk via a physical connection (e.g., USB). When loading software, the kiosk would extend the same measurement into both TPMs. This would allow both the mobile device and kiosk supervisor to obtain a quote from a TPM with a known AIK. This approach would require both driver and hardware support. Molina et al. [20] also propose the use of a mobile TPM. In their architecture, the core root of trust of the mobile TPM is measured in the permanent TPM. We envision that the two TPMs act independently and in parallel.

Alternatively, we could bind a secure channel (e.g., SSL or IPSec) to the signing key of the TPM purporting to be in the local kiosk [12]. After receiving a signed attestation from that TPM, we know that the software stack running on the other end of the secure tunnel is trustworthy. At this point, the phone can generate a secret, display it on the phone's display, and transmit the secret over the secure tunnel. The software on the other end of the tunnel may also display the secret. The user must then compare the secret shown on the display of her phone to that displayed on the kiosk. If they match, we can assume that the TPM that is bound to the secure tunnel is located within the local kiosk. Since the phone verifies the trustworthiness of the kiosk software prior to transmitting the secret, we can assume that the kiosk will not willingly give away the secret.

6.2 Reboot-between-Attestations Attack

A kiosk could reboot and run malicious software after attesting to its software integrity but before the user reveals personal data. The malicious software could then misuse the personal data. Even if the user or owner repeats the attestation request, a time window would remain during which the kiosk could reboot into and out of malicious software without being detected. The problem is not to be confused with the problem where, between reboots the system could be in a malicious state and violate the integrity or confidentiality of the VM data. This is countered by using encrypted file systems (see Section 4.3.6). The problem is that the user sitting in front of the kiosk would inadvertently enter potentially confidential data into a malicious system or accept counterfeit output from a malicious system during such a phase. To solve this problem we need to ensure that the kiosk has not rebooted between the time of attestation and the time of use by the user.

We have submitted to the Trusted Computing Group several extensions to current TPM standards that would enable remote parties to detect reboots between attestations [13]. One possibility is to incorporate a monotonically increasing reboot counter in the TPM quote structure that is signed inside the TPM. This way, if the reboot counter value in the quote of two succeeding attestations is the same, we are sure that the difference in the measurements describes all new software that was loaded between those attestations. Any reboot would lead to an increase in the counter value, detectable during the verification of the quote in the second attestation.

Finding a good balance between the overhead of frequent attestation (computing the quote signature inside the hardware TPM takes a comparably long time) and the time during which a system could be compromised unnoticedly is difficult. This problem is not specific to the kiosk scenario, and must be resolved to enable the safe use of remote attestation in general.

6.3 Boot Prior to User's Arrival

In Section 4.3.4, we require that the kiosk reboot as part of the trust-establishment process. As the TPM provides only load-time guarantees of integrity, this approach offers the greatest degree of security by minimizing the time between measurement and use. However, this step is extremely time-consuming, and will likely be unacceptable when the kiosk is used for short transactions.

In such situations, 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 7–9 of Figure 4. 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 has been running for longer. Currently, the TPM does not provide a means of securely determining when kiosk booted—this functionality would allow the phone to place a bound on the uptime that it considers to be secure.

The reboot in Step 16 must still occur to ensure that any sensitive data stored unencrypted in volatile memory is erased. The time consumed by this reboot is less critical because it does not occur while the user is waiting.

7 Related Work

Surie et al. [30] propose Trust-Sniffer as a means of establishing trust in transient hardware (which we refer to as a kiosk). Trust-Sniffer consists of a small software root of trust that the user provides via a USB flash drive. The kiosk boots from the flash drive, and the software root of trust validates the operating system present on the kiosk before booting it. Trust-Sniffer uses IMA to measure software before it is loaded, and will only allow the load to continue if the software is present in the measurement database found on the flash drive. Trust-Sniffer emphasizes the security gains possible from a simple low-cost flash drive. In contrast, the user in our system is equipped with a much more powerful smartphone, which allows us to achieve stronger security guarantees. Specifically, Trust-Sniffer must trust the BIOS and is susceptible to the attack in which the entire Trust-Sniffer environment is booted within a rogue virtual machine monitor. Our system establishes a dynamic root of trust to exclude the BIOS from the trusted computing base and uses the TPM to detect the rogue virtual machine monitor attack. Additionally, our system emphasizes the ability to load a personalized computing environment on the kiosk in a way that provides security guarantees to both the user and kiosk owner.

Sinclair and Smith [28] introduce PorKI, which allows the user to perform private key operations on machines of varying trustworthiness. The user's mobile device maintains the user's private key, and uses it to temporarily delegate authority to a short-lived keypair known to the machine in question. The trustworthiness of this machine is explicitly encoded in a credential signed by the system administrator, which is factored into the decision of a remote party to trust operations that use the temporary private key. In contrast, our system verifies the load-time integrity of the software running on the machine, which we feel provides a stronger security guarantee. Furthermore, our system considers the confidentiality of general user data (e.g., a VM) in addition to private keys.

Asokan et al. [4] investigated the problem of authenticating public terminals with trusted servers, i.e., those of credit-card companies. They propose authentication schemes for establishing a secure channel between a kiosk and a server to prevent information leakage through the network. Our work extends theirs by verifying the software loaded on the kiosk. Our goal is to avoid revealing private information to possibly corrupted software.

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 contrast, we adopt the *trusted boot* approach, in which the boot process merely measures each software component as it is loaded. Compromised software will be detected when the machine uses the TPM to attest to its software stack.

Brands et al. [7] propose *distance-bounding procotols* where the time delay between two communication parties is used to bound the physical distance of the two parties. The approach is not accurate enough to distinguish the cheating party from the honest party when they are physically close to each other, as in the kiosk scenario.

We utilize the work of McCune et al. [19] to securely obtain the public key of a wireless device (a kiosk in our case). Saxena et al. [25] suggest optimizations that reduce the bandwidth necessary of the visual channel to the point where a blinking LED may be used in lieu of a barcode. Controlling this LED through software, however, enables potential software-based attacks not possible with a fixed barcode. Stajano and Anderson [29] accomplish similar objectives through the use of physical contacts. While less susceptible to physical tampering, their approach requires hardware that is not commonly available on commodity mobile devices.

For devices that communicate exclusively over a wireless medium, probabilistic channel hopping [2] greatly increases the number of channels that must be relayed to perform a man-in-the-middle attack. However, this solution does not work as well in systems where one party (e.g., a kiosk) has vastly greater resources than another (e.g., a mobile phone).

Naor and Pinkas [21] prove that data may be securely transmitted to a human equipped with nothing more than a secret, pre-printed transparency. We are also concerned with protecting the secrecy of any data that the user provides to the kiosk.

In this paper, we assume that the user's personal device is trustworthy. Pfitzmann et al. [23] explore the many challenges of using a mobile device in a security-sensitive setting. Some of the issues they disscuss are relevant to the security of the mobile device in our kiosk scenario.

8 Conclusion

We present 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 process leverages several emerging technologies, namely the Trusted Platform Module, 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 demonstrate the viability of our approach through implementation, and show how our system supports the resumption of a user's personal virtual machine on the kiosk.

References

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

[2] A. Alkassar, A.-R. Sadeghi, and C. Stüble. Secure object identification - or: Solving the chess grandmaster problem. In *Proc. of New Security Paradigm Workshow (NSPW)*, 2003.

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

[4] N. Asokan, H. Debar, M. Steiner, and M. Waidner. Authenticating Public Terminals. *CompNet*, 31(8), 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*, volume 3650 of *Lecture Notes in Computer Science*, pages 431–445, September 2005. An extended version of this paper appears as CMU Computer Science Department Tech Report 05-111.

[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] S. Brands and D. Chaum. Distance-bounding protocols (extended abstract). In *Theory and Application of Cryptographic Techniques*, pages 344–359, 1993.

[8] E. Brickell, J. Camenisch, and L. Chen. Direct Anonymous Attestation. In *Proc. of ACM Conference on Computer and Communications Security*, 2004.

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

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

[11] S. Garriss, R. Cáceres, S. Berger, R. Sailer, L. van Doorn, and X. Zhang. Towards Trustworthy Kiosk Computing. In *Proc. of 8th IEEE Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2007.

[12] K. Goldman, R. Perez, and R. Sailer. Linking Remote Attestation to Secure Tunnel Endpoints. In *Proc. of 1st ACM Workshop on Scalable Trusted Computing*, 2006.

[13] K. Goldman and R. Sailer. Making reboot between TPM attestations visible. TCG standards discussions, 2006.

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

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

[16] B. Kauer. OSLO - The Open Secure LOader. http://os.inf.tu-dresden.de/ kauer/oslo/.

[17] 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 2006 IEEE Symposium on Security and Privacy*, 2006.

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

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

[20] J. Molina, H. Lee, S. Lee, and Z. Song. A Mobile Trusted Platform Module (mTPM) Architecture. In *Proc. of 2nd Workshop on Advances in Trusted Computing (WATC06)*, 2006.

[21] M. Naor and B. Pinkas. Visual authentication and identification. *Lecture Notes in Computer Science*, 1294, 1997.

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

[23] A. Pfitzmann, B. Pfitzmann, M. Schunter, and M. Waidner. Trusting mobile user devices and security modules. *IEEE Computer*, 30(2):61–68, February 1997.

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

[25] N. Saxena, J.-E. Ekberg, K. Kostiainen, and N. Asokan. Secure Device Pairing Based on a Visual Channel (Extended Abstract). In *Proc. of IEEE Symposium on Security and Privacy*, 2006.

[26] 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.

[27] 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 ACM Symposium on Operating Systems Principles*, 2005.

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

[29] F. Stajano and R. Anderson. The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks. In *Security Protocols Workshop*, 1999.

[30] A. Surie, A. Perrig, M. Satyanarayanan, and D. Farber. Rapid Trust Establishment for Transient Use of Unmanaged Hardware. In *Technical Report CMU-CS-06-176*, 2006.